
CONFIDENTIAL

The Amiga CD³² Developer
Notes are CONFIDENTIAL.
THE INFORMATION
CONTAINED HEREIN IS
PRELIMINARY, PROPRIETARY,
AND SUBJECT TO CHANGE
WITHOUT NOTICE.

► Copyright

Copyright 1993 Commodore-Amiga, Inc. All rights reserved. The materials used herein have been reproduced with the permission of the authors indicated. Use or reproduction of such materials shall be in accordance with the authors' instructions. Commodore and Amiga are registered trademarks of Commodore Electronics Ltd. and Commodore-Amiga, Inc. respectively. This document may also contain reference to other trademarks and registered trademarks for the various products listed which are believed to belong to the sources associated therewith.

► Warning

The information contained herein is subject to change without notice. Commodore specifically does not make any endorsement or representation with respect to the use, results, or performance of the information (including without limitation its capabilities, appropriateness, reliability, currentness or availability).

► Disclaimer

This information is provided "as is" without warranty of any kind, either express or implied. The entire risk as to the use of this information is assumed by the user. In no event will Commodore or its affiliated companies be liable for any damages, direct, indirect, incidental, special or consequential, resulting from any defect in the information, even if advised of the possibility of such damages.

► Revision

This is revision 3 of this document, August 30, 1993

CONFIDENTIAL

THE INFORMATION CONTAINED HEREIN IS
UNCLASSIFIED EXCEPT WHERE SHOWN
OTHERWISE

1. Copyright

This document is copyrighted by the U.S. Government. All rights are reserved. No part of this document may be reproduced or transmitted in any form or by any means electronic or mechanical, including photocopying and recording, or by any information storage or retrieval system, except as may be permitted in writing by the U.S. Government.

2. Release

This document is released under the terms of the U.S. Government's policy of maximum availability of information to the public. It is hereby authorized for release to the public without limitation.

3. Distribution

This document is distributed to all U.S. Government agencies and departments. It is also available to the public upon request. The U.S. Government is not responsible for the use or misuse of the information contained herein.

4. Revision

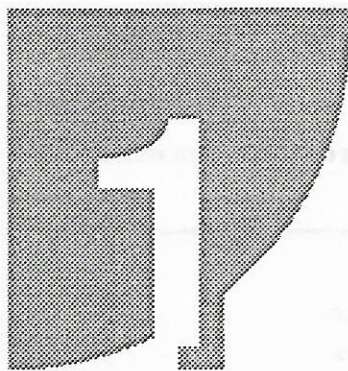
This document is revised as necessary to reflect changes in the U.S. Government's policy or practice. The U.S. Government is not responsible for the use or misuse of the information contained herein.

Contents

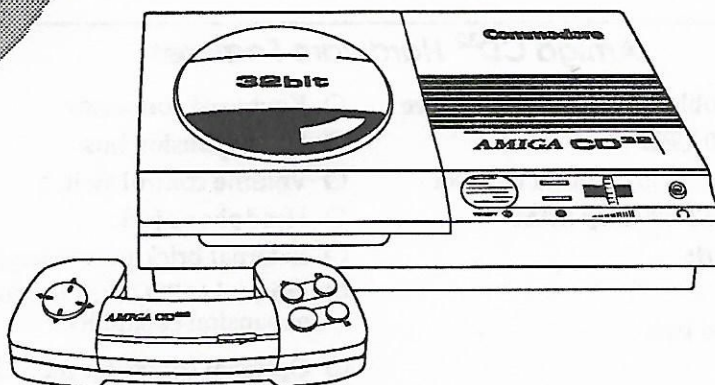
Chapter:	1	Introduction	1
	2	Producing Titles	5
	3	Writing OS-Friendly Games	21
	4	Includes and Autodocs	55
	5	Expansion Connectors	93
Appendix:	A	Questions and Answers	97

Contents

Chapter 1	Introduction	1
Chapter 2	Processing Files	5
Chapter 3	Writing OS-managed Datasets	21
Chapter 4	Includes and Aliases	53
Chapter 5	Expansion Conventions	93
Appendix A	Questions and Answers	11



Introduction



An exciting product has emerged from Commodore engineering. It combines the graphics excellence of the Amiga computer family with the megastorage and audio purity of the CD-ROM, making it an instant force for all to reckon with. There's a new standard in the video game world and its name is the Amiga CD³² game system.

The Amiga CD³² is an entry level, 32-bit CD game console based on the Amiga AA chip set and a 68EC020 microprocessor. The AA chips, the latest of the custom chip sets that characterize the Amiga family, provides a palette of 16 million colors, a dazzling variety of display modes and four-voice, 8-bit, stereo audio.

This is a machine that can transport players into the vortex of an alien hyperspace or the depths of a dank dungeon far below an evil warlock's castle. You'll be able to provide your starship captains with full-motion video on their viewscreens and klaxons so real, the neighbors will go to general quarters. The Amiga CD³² is a product ready to do battle in the fiercely competitive game market, but it can't go it alone, it needs a software creator. It needs you.

In this document, we present sections on the Amiga CD³² hardware and system software, title production, game programming techniques, expansion hardware, and reference documentation.

Section Outlines

- | | | |
|----------|----------------------------------|---|
| 1 | Introduction | What Amiga CD ³² is, why it exists, who will buy it and who is the competition. |
| 2 | Producing Titles | Development tools, porting existing Amiga titles, CD mastering and duplication, and distribution information. |
| 3 | Writing OS-Friendly Games | The CD ³² libraries, file system and game programming techniques. |
| 4 | Includes and Autodocs | Reference material about the Amiga CD ³² software libraries and file system. |
| 5 | Expansion Connectors | System expansion connectors and other hardware. |
| A | Questions and Answers | Answers to commonly asked questions. |

► Amiga CD³² Hardware Summary

The Amiga CD³² is a game system first, then a computer. Its basic hardware configuration reflects this. The table below summarizes the hardware features.

Amiga CD³² Hardware Features

- | | |
|---|--|
| ☐ Top loading double speed CD-ROM drive | ☐ Keyboard connector |
| ☐ 14MHz 68EC020 CPU | ☐ Full expansion bus |
| ☐ Amiga AA graphics and sound chip set | ☐ Volume control switch |
| ☐ 2 Megabytes of 32-bit Chip RAM | ☐ Headphone jack |
| ☐ Two joystick ports | ☐ External brick power supply |
| ☐ S-video jack | ☐ Internal MPEG Full Motion Video (FMV) expansion capability |
| ☐ Composite video jack | ☐ Optional computer module |
| ☐ RF output jack | ☐ Multiple session disc capability |
| ☐ Stereo audio jacks | |

► The Amiga CD³² Motherboard

The motherboard for the Amiga CD³² is roughly 6" by 12" and is a four layer board. On the main PCB is a 14MHz 68EC020, the AA chipset, an ASIC called Akiko, a DAC for playing standard CD audio, and a small amount of EEPROM.

Akiko is a 160-pin PQFP surface-mounted device containing most of the remainder of the logic necessary in the Amiga CD³². It includes the CD-ROM control logic and the system timers. The EEPROM is 8K bits, and designed for storing settings, high scores, bookmarks, etc., on the Amiga CD³².

In addition to the AA chips, Akiko, the CPU and the EEPROM, the main board contains 2MB of DRAM as Chip RAM. A PAL or NTSC modulator is also included on the main board.

The Compact Disc DAC is a standard 18-bit, 8x oversampling DAC. Along with the superb audio quality of the DAC, the Amiga CD³² expansion connector has digital audio input capability in order to mix an external source such as MPEG audio with the CD audio and Amiga audio.

► The Amiga AA Chip Set

The Amiga AA chip set is the latest in the Amiga family of custom chips. It provides stunning graphics and animation capabilities, a huge color palette and multiple display modes. Throughout the history of the Amiga, the individual chips of each chip set have had colorful names beginning with the trio of Portia, Agnus and Daphne in the original chip set up to the latest trio of Paula, Alice and Lisa contained in the Amiga CD³² system.

- ☐ Paula controls the I/O ports, potentiometer inputs, the audio hardware and interrupt control.
- ☐ Alice is the DMA controller. It generates DMA addresses, controls Chip RAM access, video timing and includes the blitter and copper (co-processor).
- ☐ Lisa is the video data chip. It provides the 16-million color palette, playfield scrolling, sprite control, joystick and Genlock support.

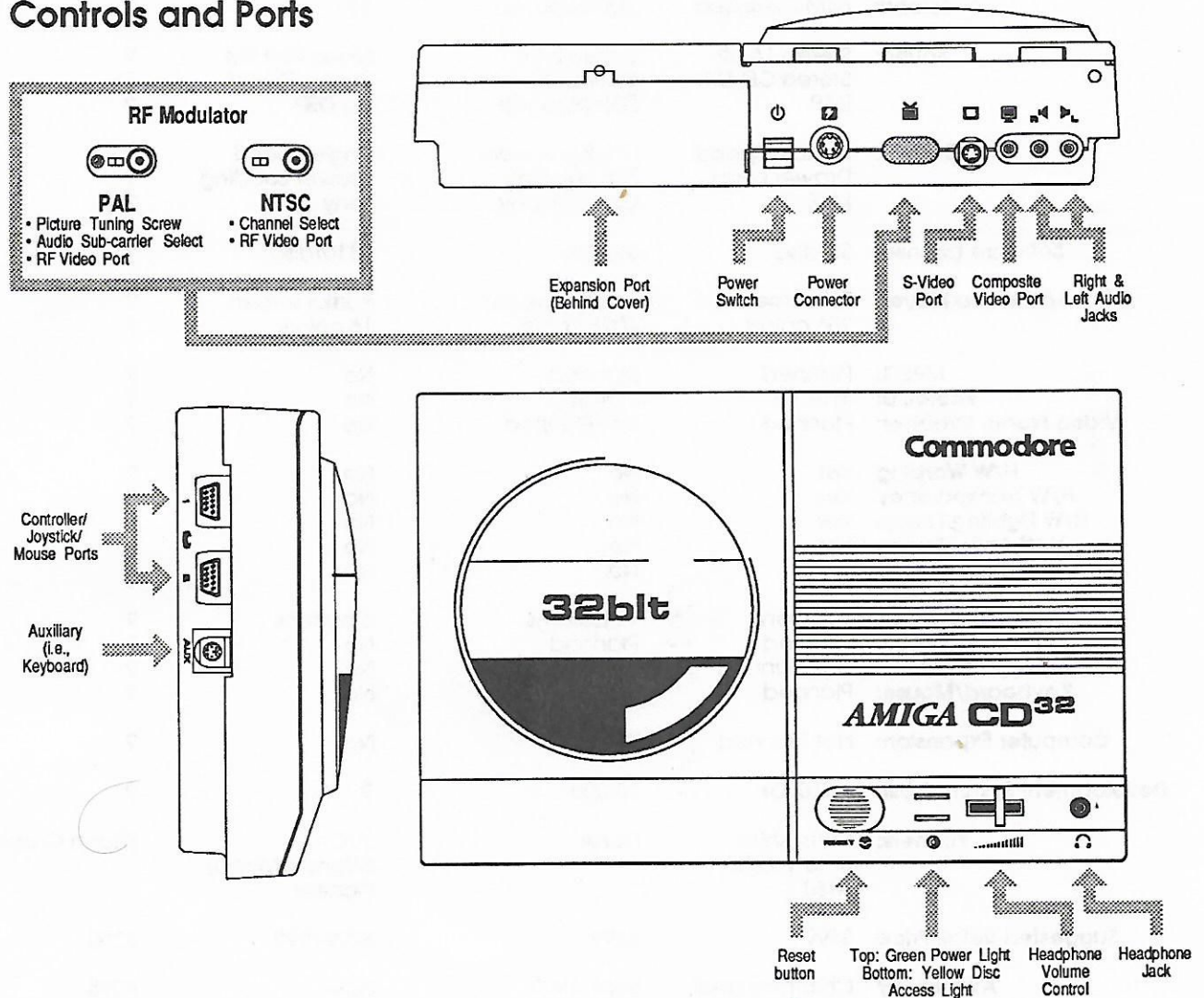
AA Chip Set Features

The AA chip set has a number of features that are ideal for games.

- 256 colors out of 16 million in all resolutions
- 32-bit wide Chip RAM and support for page mode DRAMs
- 8-bit HAM mode support (near true color display)
- Sprites are up to 64 bits wide
- Sprites can be displayed in the borders
- Attached sprites are available in all resolutions
- Dual 4-bitplane playfields
- Hardware scan-doubling support
- Hardware compatibility mode with the older Amiga ECS chip set
- Four-voice, 8 bit audio with variable sample rate

The AA chip set supports 15kHz and 31kHz display modes, but because Amiga CD³² will primarily be attached to a television set, the 31kHz modes are not used. These video modes can be used if the optional computer module is added with the RGB interface.

Controls and Ports



Amiga CD³² Comparison with other platforms

	<u>3DO</u>	<u>Amiga CD³²</u>	<u>SegaCD</u>	<u>NintendoCD</u>
CPU/Speed:	ARM60/12 mHz	68EC020/14 mHz	2x68000/12 mHz	RISC processor/?
Bits:	32 bit	32 bit	16 bit	32 bit
MIPS:	6 MIPS	3.5 MIPS	0.3 MIPS	?
Chip RAM:	1M VRAM	2M DRAM	64KB VRAM	?
Fast RAM:	1M DRAM	None	64KB SRAM	
Non Volatile RAM:	1KB	8KB	Yes, ?KB	
Custom Chips:	Two animation Processors Video Processor DSP, DMA Engine Exp. Controller	I/O ports, audio and Interrupt controller DMA controller Video data controller CD-ROM controller	2 custom chips In base 3 custom chips In CD	?
Animation CELS:	Yes (100s)	8 Sprites (64 bit) & Bobs	80 Sprites (32 bit)	?
Video Modes:	640x400, 15kHz	640x400, 15kHz	320x200, 15kHz	?
Colors:	256/32768	256,000/16 Million	64/512 colors	?
Speed:	64M Pixels/sec	7M Pixels/sec	??	?
Sound:	Stereo 16 bit Stereo CD-DA DSP	Stereo 8 bit Stereo CD-DA DSP planned	Mono 8bit FM Stereo CD-DA No DSP	? ? ?
CD-ROM:	Double Speed Drawer Load MKE	Double Speed Top Loading Chinon/Sony	Single Speed Drawer Loading Sony	? ? ?
Software License:	\$3/disc	\$3/disc	~\$10/disc	?
S/W Video Player:	Full screen 256 colors	Partial screen 4096 colors	Partial screen 16 colors	? ?
MPEG:	Planned	Planned	No	?
PhotoCD:	In ROM	Planned	No	?
Video Frame Grabber:	Planned	Not Planned	No	?
H/W Warping	Yes	No	No	?
H/W Transparency	Yes	No	No	?
H/W Lighting Effects	Yes	No	No	?
H/W Anti-Aliasing	Yes	No	No	?
H/W Texture Mapping	Yes	No	No	?
Game Controller:	8 buttons	11 buttons	8 buttons	?
Parallel/Serial:	Planned	Planned	No	?
Floppy/Hard Drive:	Not Planned	Planned	No	?
Keyboard/Mouse:	Planned	Planned	No	?
Computer Expansion:	Not Planned	Yes	No	?
Development System Cost:	\$10,000+	\$3,000	?	?
Partners:	Matsushita Time Warner AT&T	None	JVC (WonderMega) Pioneer	Silicon Graphics
Suggested Retail Price	\$699	\$399	\$299+\$99	\$250
Availability	Christmas goal	Sept. 1993	Now	1995

2

Producing Titles

Producing titles for the CD game machine platform is a little different than the method you might use for floppy-based titles. The basic steps are as follows:

1. Get a development system (A4000 with a hard drive).
2. Develop or port game.
3. Create an ISO image of the game with ISOCd. Test with SimCD on an A4000.
4. Create the gold (master) disc from the ISO image.
5. Test the gold disc on a prototype CD game machine.
6. Repeat steps 2-5 until ready for duplication.

Development Systems

An Amiga 4000 with an '040 CPU is a good development platform. The A4000 has the AA chip set so the underlying hardware will be the same as the game machine. The '040 processor means that you will be able to use debugging tools such as Enforcer that require an MMU.

You should use the MapROM command with the V40 developer Kickstart release to make the A4000 more like the game machine operating system. MapROM also allows you to upgrade as new V40 Kickstarts are released (no ROMs or EPROMs required).

The IDE disk drive bundled with the A4000 gives a reasonable approximation of CD-ROM speeds. For faster, larger hard drives, use a 4091 or 2091 SCSI controller and a SCSI drive. (If you use the 2091 controller board, it must have the 7.0 ROM set to work properly with the A4000.)

Since the CD game machine has a 68EC020 processor, you may also want to have an Amiga 1200 for accurate performance testing of your product prior to mastering. The A1200 has an '020 processor and the AA chip set just like the Amiga CD³². However be aware that MMU-based debugging tools such as Enforcer will not work with the '020 processor.

The Amiga CD³² game machine is shipped with Chip RAM only. For testing, you should use either the NoFastMem command (do this early in your startup-sequence) or remove all Fast RAM from your test system so that your code loads into Chip memory only. Keep in mind that Amiga CD³² memory can be expanded so your code should be able to run with Fast memory too.

Testing with Amiga CD³² and the Debug Board

Amiga CD³² development systems are available in limited quantities that allow you to test gold (master) discs. Development systems also include a game controller and a special development plug-in card that has serial, parallel, RGB, IDE and floppy connectors. This plug-in card is officially known as the "Debug Board".

With the debug board you can run and test your code off of an IDE hard disk drive (or floppy disk). The debug board plugs into the expansion edge card connector at the rear of the game machine motherboard. The operation of the ports and their physical connectors are identical to the standard Amiga I/O ports.

Floppy drive

Supports any standard Amiga drive (A1010, A1011). Do not plug or unplug drives from this port with the power on. The 8520s connected to this port are very static sensitive and can be easily damaged this way. This port has the highest boot priority.

Parallel port

Identical to other Amiga parallel ports.

Serial port

Identical to other Amiga serial ports except that Ring Indicate is not supported. This is useful for serial port debugging with the debug.library and other tools such as Enforcer.

IDE Interface

This can support any size IDE hard drive with a 40-pin interface connector. A separate power supply must be provided for the hard drive. The connector is not keyed so carefully note the pin 1 orientation. This interface is memory-mapped I/O, not DMA, so it is relatively slow. The hardware implementation on the debug board is even slower than an A1200 IDE interface so you may notice some exceptionally long boot times from hard disk.

Emulator

The socket on the debug board labelled U1 allows you to connect a 68020 In-Circuit-Emulator pod to the system. The 68020 ICEs can be used in place of the 68EC020 without any problems. To enable the emulator, short the two pads of XJ1 on the debug board together. (This asserts a bus request signal to the 68EC020 processor on the main board disabling it.)

Patch RAM

This optional RAM may or may not be installed on your board (depending on its vintage). This is 512K bytes of static RAM located at \$F0 0000 used for patching the operating system firmware. It is not needed for normal debugging operation.

RGB video

This is analog RGB. The connector may or may not be present on your board (depending on its vintage) but the pinout is identical to Amiga video, so a cable could be hard-wired to the board. If this is too much trouble, use the S-video signal from the main PCB; it is nearly as good as the analog RGB.

Note that if you have one of the early prototype systems, the debug board is required in order for the system to operate. The prototype systems cannot boot without the 8520 CIAs that are part of the debug board. Installation of the debug board should be done with caution. It requires a good deal of force to insert the board and the pins inside the connector are fragile. Once bent, the pins cannot be straightened.

Development Tools: ISOCD, SimCD and OPTCD

There are three development tools that can help you in your effort to make CD titles:

- ISOCD** - Creates an ISO-9660 image file suitable for making the gold (master) disc.
Prepares a partition for use with SimCD.
- SimCD** - Emulates the CD-ROM drive (CD0:) from a partition prepared with ISOCD. This allows you to test the integrity of your ISO image without having to cut a gold (master) disc.
- OPTCD** - Optimizes the arrangement of data on a CD to minimize seek times and improve performance.

ISOCD, SimCD and OPTCD are all part of the ISO Tools Disk available to any licensed Amiga CD³² developer. With ISOCD, SimCD and OPTCD you can create and test the integrity of ISO images on hard disk before taking the time and expense to cut a gold (master) disc.

ISOCD has two purposes. It prepares a partition for use with SimCD and it also can produce the ISO-9660 image file you need in order to create the gold disc. The basic steps to using ISOCD are as follows:

1. Set up a partition of the appropriate size on hard disk named CDN:. (Do not call it CD0:. That name is reserved.)
2. Run ISOCD. Click on the Source gadget to specify the source directory containing your program and data files.
3. Click on the Image gadget to specify the target partition to use for the ISO image. Only specify a file name for the image if you are ready to cut a gold (master) disc.
4. Click on the Examine gadget. A list of files is displayed. (Do not use the "Add Empty Space" feature with V1.00; it is broken. This option works in 1.03 and later.)
5. Click on the Build gadget. Exit.

SimCD sets up a simulated CD-ROM drive named CD0: using the ISO image set up previously with ISOCD. With SimCD, you can check to see if the ISO-9660 version of your application will work. There is no need to cut a disc. The steps to using SimCD are:

1. Make a directory called devs:Local. Copy devs/local/cdtv.device from the ISO Tools disk into devs:Local.
2. Make a directory called l:Local. Copy l/local/cdfs from the ISO Tools disk into l:Local.
3. Make 2 assignments. Assign devs: devs:Local add. Assign l: l:Local add.
4. Load SimCD. Under Configuration, select CDFS and cdtv.device only.
5. Set "Sim Device" to CDN: and hit the Simulate button.
6. SimCD sets up a simulated CD0: drive using the ISO image on CDN:. Move to the directory CD0: and run your startup-sequence or executable.



Warning:

SimCD emulates the older `cdtv.device` driver used in Commodore CDTV systems. Commodore is working to upgrade this important tool so that it emulates the new `cd.device` driver used in the CD game machine. Until then, be aware that SimCD cannot be used to test code that directly calls the `cd.device` or uses CDXL reads. Also, SimCD emulates the slower speeds of the CD-ROM drive used with CDTV. Your title on disc will be faster.

Once you have tested your application in its ISO-9660 version with ISOCD and SimCD, you are ready to create the gold (master) disc.

▶ Trademark File

In order to get a title to boot on an Amiga CD³² console, a special Amiga CD³² trademark file must be included directly after the PVD sectors. CDTV used a similar trademark file either right after the PVD sectors or in a normal file in the root of the CD directory structure. This is not the case with AmigaCD³², the trademark file *must* always be included after the PVD sectors.

The Amiga CD³² trademark file differs from the CDTV one. The Amiga CD³² system relies on the differences in the trademark files to determine if a title is a CDTV title or an Amiga CD³² title. Contact CATS for information on obtaining the trademark file if you plan on generating your own ISO image. In order to ensure the trademark file gets placed directly after the PVD sectors, do not simply copy it to your CD's directory. Instead make sure that the trademark file is in the directory you are currently CD'd to (probably where ISOCD is located) when creating your ISO image.

Note that use of the trademark file requires an Amiga CD³² license agreement with Commodore. Contact CATS for licensing information.

The Amiga CD³² system is compatible with some older CDTV titles. The type of trademark file on a CD determines the environment in which the title is started. If a CDTV trademark file is found, the title is booted in ECS mode. If an Amiga CD³² trademark is found, the title is booted directly in AA mode. In addition, many system patches are applied on a per-title basis when booting CDTV titles in order to keep them working on the Amiga CD³².

▶ Creating the Gold Disc

CD manufacturing companies make a gold (master) disc before they produce your title in large quantities. This step is known as pre-mastering and can usually be performed by the same company that produces the discs in quantity.

For a limited time, Commodore will help you create the gold disc. This service is provided to shorten the amount of time it takes to bring your product to market. (It is OK to use a pre-mastering service of your own choosing if you prefer.) To take advantage of this service, send the ISO image of your application on DAT tape to the Maidenhead office in the U.K., care of David Pocock. A gold disc will be created and, if it works, an additional disc will be created for examination by Commodore, West Chester. In the U.S., send the DAT tape directly to the West Chester office care of Larry Feldman.

The gold disc will be returned to you as quickly as possible so that you can test it on a CD game machine prior to manufacturing it in quantity. When the gold disc is created, any protection bits or script bits you have set for your files will be lost since these are not supported by the ISO-9660 standard. Be sure to avoid counting on the presence of these bits in application code.

There are a variety of ways to convey your ISO image to the pre-mastering service. Depending on the vendor, you might send off a hard disk, a DAT tape containing a copy of the ISO, 1/2" U-Matic video tape — some vendors even accept floppies. If you use Commodore for your pre-mastering service, try to provide your ISO image on DAT tape, if possible. Other formats will take longer. If you do not supply an ISO image on DAT tape, be sure to include the volume name you want to use for the CD-ROM disc. If your title also includes CD audio, you should try to provide this on separate media, preferably a DAT tape.

■ Manufacturing CD-ROMs

While it is possible to create a "master-stamper" directly from the gold (master) disc, most replication centers prefer to receive a 9-track tape. This master tape is an ANSI-labelled, 9-track, 1/2" tape containing the ISO-9660 image of your code and data. This can be prepared by you or by the service that created the gold disc.

The CD manufacturer will transfer the tape to a laser beam recording system which creates the master stamper. The stamper presses CDs from molten polycarbonate which is subsequently coated with aluminum and a protectant. Labels are added and the disc inserted into a jewel case or other packaging.

In the U.K., the recommended vendors for volume production of your CD-ROM based application are:

SonoPress

26/27 Conduit St.
London W1R 9TA
Voice: 71 499 6813
Fax: 71 493 7244

Nimbus

Wyastone Leys
Monmouth
Gwent NP5 3SR
Voice: 600 890 682
FAX: 600 890 779

Pricing is subject to negotiation and volume but a rough rule of thumb is:

Mastering: £ 900
Set up: £ 200
Per disc: £ 0.90 to 1.50

There are many other vendors you can choose to work with. Here is a brief list to get you started in your search:

3M Optical Recording Department

3M Center 223-5S-01
St. Paul, MN 55144-1000
Telephone (612) 736-5399
Fax (612) 733-0158
Sales contact: Don Winklepleck, (603) 595 0391
Technical contact: Andy Axelsen, (800) 336 3636
Fax: (715) 235 0500
Territories served: North and South America, Europe, Far East
Preferred source format: 8mm Exabyte, 4 mm DAT, 9 Track, MO, Hard Disk, CDWO, 3480, DC6150 (QIC)
One-off service available: Call
Terms of business: Net 30 days
Pre-mastering and CDTV ISO building: Call

Advanced Media Group, Ltd.

P.O. Box 1623
Lancaster, PA 17603
Telephone: (717) 392-6533
Fax: (717) 392-0532
Sales contact: Stan Caterbone
Technical contact: Stan Caterbone/Mike Hess
Territories served: North and South America, Europe, Far East
Preferred source format: Tape, hard disk, any media accepted upon prior approval
One-off service: No
Terms of Business: Upon invoice Net 30 with prior approval
Pre-mastering and CDTV ISO building: No

Americ Disc Inc.

2525 Canadien
Drummondville, Quebec
Canada J2B8A9
Telephone: (819) 474-2655
Fax: (819) 474-2870
Sales contact: A. Frank Johansen
Technical contact: Peter Frame
Territories served: Canada, USA, South America
Preferred source format: 8mm Exabyte, 9 track tape, CD-ROM
One-off service: No
Terms of Business: Net 30 days on credit approval
Pre-mastering and CDTV ISO building: No

Attica Cybernetics Ltd.

Unit 2 Kings Meadow
Ferry Hinksey Road
Oxford, England OX2 0DP
Telephone: 44-865-791-346
Fax: 44-865-794-561
Sales contact: Gill Diskson
Technical contact: Ian Ellison
Territories served: Worldwide, from UK office
Preferred source format: AmigaDOS file on SCSI drive. Other options available
One-off service: Call
Terms of Business: Payment in advance; credit for UK companies by arrangement
Pre-mastering and CDTV ISO building: Call

Clarinet Systems Ltd.

White Hart House
London Road
Blackwater, Camberley, Surrey, UK
Telephone: 44-276-600-398
Fax: 44-276-600-592
Sales contact: Stephen Schoiefield
Technical contact: Chris Simmonds
Territories served: UK, Europe
Preferred source format: Hard disk, DAT, floppy, Exabyte, 1/2 inch tape
One-off service: Available
Pre-mastering and CDTV ISO building: Available

Digipress. U.S.

2516 River Bend Drive
Louisville, KY 40206
Telephone: (502) 895-0565
Contact: Dennis Oudard

Digipress. Europe

10 rue de Paris
78100 Saint-Germain-en-Laye, France
Telephone: 33-1-30-61-11-00
Contact: Marc Deflassieux

Digital Audio Disc Corporation (DADC)

1800 N. Fruitridge Ave.
Terre Haute, IN 47804
Customer service Telephone: (812) 462-8192
Customer service Fax: (812) 466-2007
Sales contact: Bob Hurley, Voice: (603) 595-4331 Fax: (603) 595-4310
Technical Contact: Cliff Brannon, Voice: (812) 462-8286, Fax: (812) 466-9125
Territories served: USA, Europe, Far East
Preferred source format: 9 track tape, MO, 8mm, 4mm, CD WO
One-off service: Available
Terms of Business: 1% 10 Net 30
Pre-mastering and CDTV ISO building: Available soon

Discovery Systems

7001 Discovery Boulevard
Dublin, OH 43017
Telephone: (614) 761-2000
Fax: (614) 741-4258
Sales Contact: Greg Tiller
Technical Contact: Alex Deak, Customer Service
Territories served: North America, Europe, Australia
Preferred Source format: 9 track or 8mm tape, call for additional options
Terms of business: 90 days net
One-off service: Available
Pre-mastering and CDTV ISO building: Yes

Disc Manufacturing, Inc., International

A Quixote Company
4905 Moores Mill Rd.
Huntsville, AL 35810
Telephone: (205) 859-9042
Fax: (205) 859-9932
Sales contact: Kim Vandenberghe
Technical contact: Shogo Karitani
Territories served: North and South America, Europe, Far East
Preferred Source format: 8mm Exabyte, Pinnacle MO, One-off CD, 9-track tape
One-off service: Call
Terms of Business: Net 30 days, on credit approval
Pre-mastering and CDTV ISO building: Call

Disc Manufacturing, Inc.. U.S.

1120 Cosby Way
Anaheim, CA 92086
Telephone: (714) 630-6700
Fax: (714) 630-1025
Sales contact: Wan Seegmiller
Technical contact: Leon Whidbee

Elektroson

Velderseweg 25
5298 Le Liempde
The Netherlands
Telephone: 31-4113-3021
Fax: 31-4113-2763
Sales and technical contact: Dr. R.C.H. Broers
Territories served: Europe
One-off service: Yes
Pre-mastering and CDTV ISO building: Yes

M.P.O.

195 Ave. Charles de Gaulle
92200 Neuilly Sur Seine
France
Telephone: 331-4722-2000
Fax: 331-4722-6077
Sales contact: Bruno d'Orgeval
Technical contact: Marc des Rieux
Territories served: USA/Canada (Disc Americ), Europe (MPO), Spain (Techno-CD)
Preferred Source Format: Video 8mm
One-off service available: Yes
Terms of business: 60 days
Pre-mastering and CDTV ISO building: Yes

Multi-Media Masters Machinery SA

Av. des Sports 42
CH - 14000 Yverdon-Les-Bains
Switzerland
Telephone: 41-24-23-71-11
Fax: 41-24-23-71-12
Sales contact: Gregory Koler
Technical contact: Albert Khoury
Territories served: Europe, USA if required
Preferred Source Format: 9-track tape, 8mm Exabyte, 1/4" U-Matic, DAT, 5.25" Optical
One-off service: Call
Terms of business: Net 30 days
Pre-mastering and CDTV ISO building: Call

Next Technology Corporation Ltd.

St. John's Innovation Center
Cambridge
Cambridgeshire CB4 4WS
U.K.
Telephone: 44-223-420-222
Fax: 44-223-420-015
Sales contact: Ian Thomas
Technical contact: Neil Critchell
Territories served: Europe
Pre-mastering and CDTV ISO building: Yes

NEXT specializes in pre-mastering and CDTV ISO building services.

Nimbus Information Systems, U.S.

SR 629, Guildford Farm
Ruckersville, VA 22968
Telephone: (804) 985-1100 or (800) 782-0778
Fax: (804) 985-4625
Sales contact: Larry Boden
Technical contact: Ernest Runyon

Nimbus Information Systems, Europe

Wyastone Leys
Monmouth
Gwent NP5 3SR
United Kingdom
Telephone: 44-600-890-682
Fax: 44-600-890-779
Sales contact: Steve Connolly
Technical Contact: Jim Orr
Territories served: USA and Europe
Preferred source format: 4mm DAT or 8mm Exabyte
One-off service: Call
Terms of business: Net 30 days, upon credit approval
Pre-mastering and CDTV ISO building: Call

On-Site CD Services

13901 Lyndie Ave.
Saratoga, CA 95070
Telephone: (408) 867-0514
Fax: (408) 867-0518
Sales contact: Rick Wittwer
Technical contact: Lance Buder
Territories served: USA
Preferred source format: 8 mm, ANSI labeled image or hard disk
One-off service : Call
Pre-mastering and CDTV ISO building: Call.

On-Site is a pre-mastering specialist, located in California. They do not replicate discs in quantity.

Optical Media Storage S.P.A.

Localit' Campo di Pilek
67100 L'Aquila, Italy
Telephone: 39-862-3311
Fax: 39-862-315366
Technical contact: Antonio Bruno
Territories served: Europe
Preferred source format: 9-track tape, hard disc, CD-WO, CD-ROM ISO image
One-off service: Call
Pre-mastering and CDTV ISO building: No

Phillips and Du Pont Optical

1409 Foulk Road, Suite 200
Wilmington, DE 19803, USA
Telephone: (302) 479-2501
Fax: (302) 479-2512
Sales contact, USA: Joe Bradley, Voice: (301) 989-9341
Technical contact: Jim Fricks, Voice: (704) 734-4211
Territories served: USA and Europe
Preferred source format: 8mm, 9-track tape, one-off CD
One-off service: available in USA
Pre-mastering and CDTV ISO building: USA only

P.D.O. offers pan-European services. Here are contact addresses and numbers in other countries:

PDO Headquarters

Building EF-2
P.O. Box 218
5600 MD Eindhoven, Netherlands
Telephone: 31-40-751120
Fax: 31-40-757866

PDO Technical Support---Europe

Klusriede 26
D-3012 Langenhagen 1, Germany
Telephone: 49-511-7306-253
Fax: 49-511-7306-694

PDO sales office for Benelux and Italy

Building EF-2
P.O. Box 218
5600 MD Eindhoven, Netherlands
Telephone: 31-40-751120
Fax: 31-40-757866

PDO sales office for the UK, Ireland and Scandinavia

Queen Anne House
11 The Green
Richmond upon Thames
Surrey TW9 1PX, UK
Telephone: 44-81-948-7368
Fax: 44-81-940-7137

PDO sales office for France and Spain

43 Avenue Marceau
F-75116 Paris, France
Telephone: 33-1-4070-1123
Fax: 33-1-4070-1126

PDO Sales office for Germany, Austria and Switzerland

Adenauerallee 32
D-2000 Hamburg 1, Germany
Telephone: 49-40-280-1391
Fax: 49-40-280-1785

Sonopress GmbH

Carl-Bertelsmann-Str. 161
D-4830 Gutersloh 100, Germany
Telephone: 49-5241-803074
Fax: 49-5241-73686
Sales contact: Dr. Reinhard Raubenheimer
Technical contact: Ulrich Graznow
Telephone: 49-5241-805250
Fax: 49-5241-75863
Territories served: all European countries, USA if required
Preferred source format: 9-track tape, Exabyte, SCSI harddisk, Audio input media Sony 1610/1630
One-off service: Call
Terms of business: Based on customer requirements
Pre-mastering service: Call

Sonopress provides pan-European services. Here are contact names and numbers in other countries.

Sonopress UK

26/27 Conduit Street
London W1R 9TA
Sales contact: Sabine Leuerer
Telephone: 44-71-499-6813
Fax: 44-71-493-7244

Sonopress France

Sales contact Madame Bornhold
Telephone: 33-1-45-636707
Fax: 33-1-43-596673

Sonopress Italy

Sales contact Dr. Paolo Montagna
Telephone: 39-2-7600-4737
Fax: 39-2-7601-5026

Disc Packaging and Graphics Standards

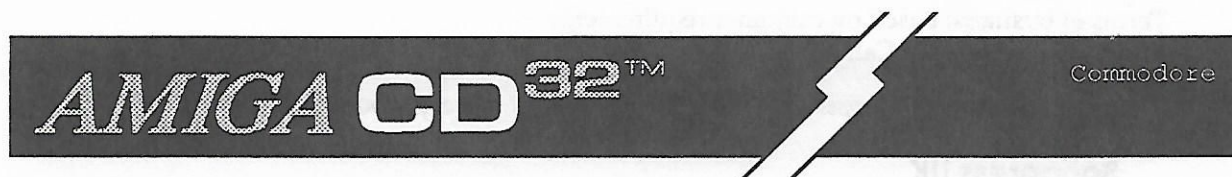
Before your title goes into production, you will need to determine its packaging. Will your discs be packaged in jewel cases? Do you plan to have any outer packaging for your disc cases? What about additional documentation, reference cards and promotional materials? All of this should be determined before hand, since most duplication houses requires packaging or packaging specs prior to or with a disc production order. Some disc duplicators provide design and printing services, relieving you of this burden.

The packaging decision is yours, all that Commodore asks is that you try to adhere to its graphics standards. This section provides the information needed to present a unified image to the public. A universal standard for the visual identity of all Amiga CD³² products.

The Logos

The Amiga CD³² logo is the foundation for the entire identity program. It consists of the of the Amiga CD³² logo type and the "TM" symbol. These elements should *never* be altered from the form shown below, except as specified in this publication. The logo's attributes are as follows:

- The Amiga CD³² logo must appear on a 100% black background.
- "Amiga" and "32" and the "TM" symbol print in PMS red 485 or equivalent. If the logo is to appear in only black & white, the previous words should be printed in a 20% black screen.
- "CD" is a white knockout (or prints in white)
- In most cases, the logo should be positioned horizontal in the upper left corner of the package.



In most instances the Commodore logo should appear in conjunction with the main Amiga CD³² logo. When this is the case, the Commodore logo should have the following attributes:

- "Commodore" is a white knockout (or prints in white)
- It is always positioned to the left of Amiga CD³², in the upper right corner of the package.
- The top of the logo lines up with the top of the "32" in the Amiga CD³² logo.
- The minimum black space between the two logos is one full length of the Commodore logo.



ALWAYS USE PHOTOMECHANICAL REPRODUCTIONS OF THE LOGOS.

NEVER...

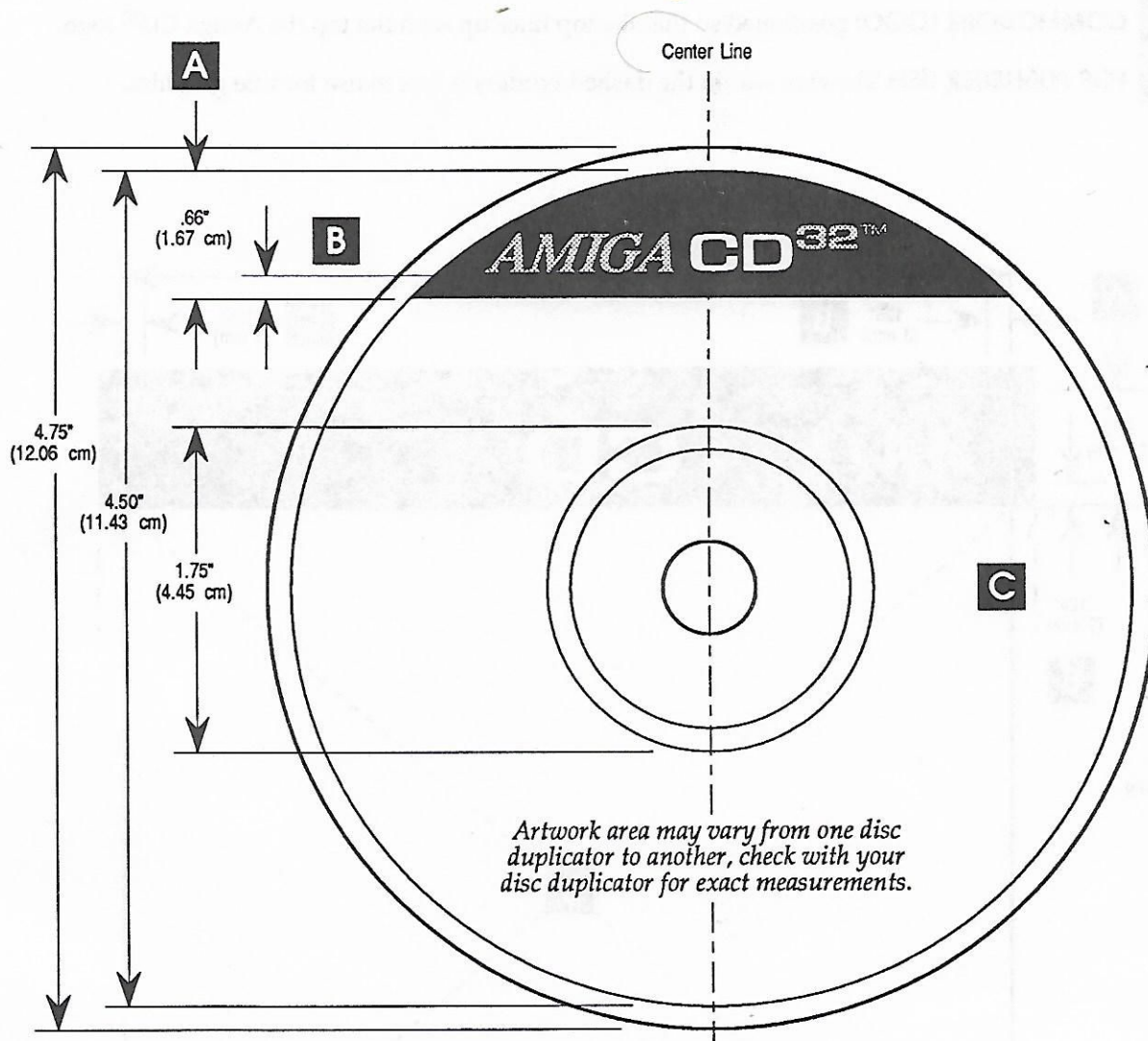
- Use photocopies of the logos.
- Attempt to redraw or re-create the logos yourself.
- Reproduce the logos from the diagrams or templates in this manual.

Reproducible "stats" can be acquired from the CATS department at Commodore. Logos are also available in Encapsulated Postscript format for use in creating packaging on desktop publishing computer systems.

Disc Graphics

Below is the proper application of graphics on the face of the disc itself. The only specification is the consistent application of the Amiga CD³² logo over a black bar.

- A TOP BAR:** A band of 100% black, .66" tall to the edge of the disc's graphic area.
- B AMIGA CD³² LOGO:** Centered horizontally within the top bar.
- C FOR PUBLISHER USE:** The remaining 4 1/2" DIA. area is free to use for title graphics



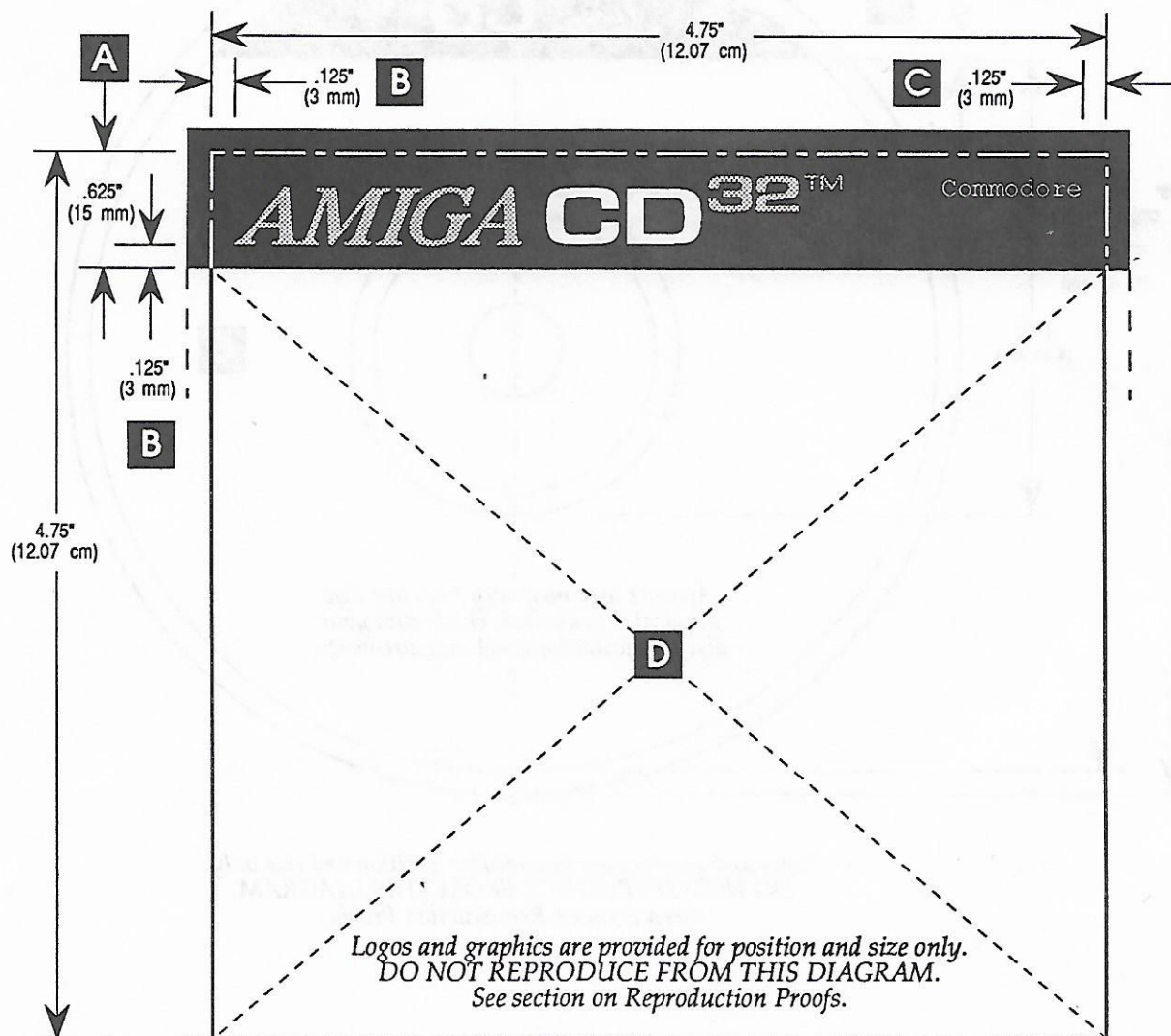
*Logos and graphics are provided for position and size only.
DO NOT REPRODUCE FROM THIS DIAGRAM.
See section on Reproduction Proofs.*

Cover Insert

The cover insert is either a single sheet or a multipage booklet placed inside the jewel case cover. For the purpose of these standards, Commodore is only concerned with the placement of its graphics on the front cover portion of the insert.

Before designing the cover insert and choosing paper stock, check with your disc duplicator for exact measurements and paper requirements. This will avoid jamming or damage during insertion.

- A TOP BAR:** A band of 100% black, 5/8" tall should bleed off the top and both sides of the card.
- B AMIGA CD³² LOGO:** Positioned as shown.
- C COMMODORE LOGO:** positioned so that the top lines up with the top the Amiga CD³² logo.
- D FOR PUBLISHER USE:** The area within the dashed borders is free to use for title graphics.

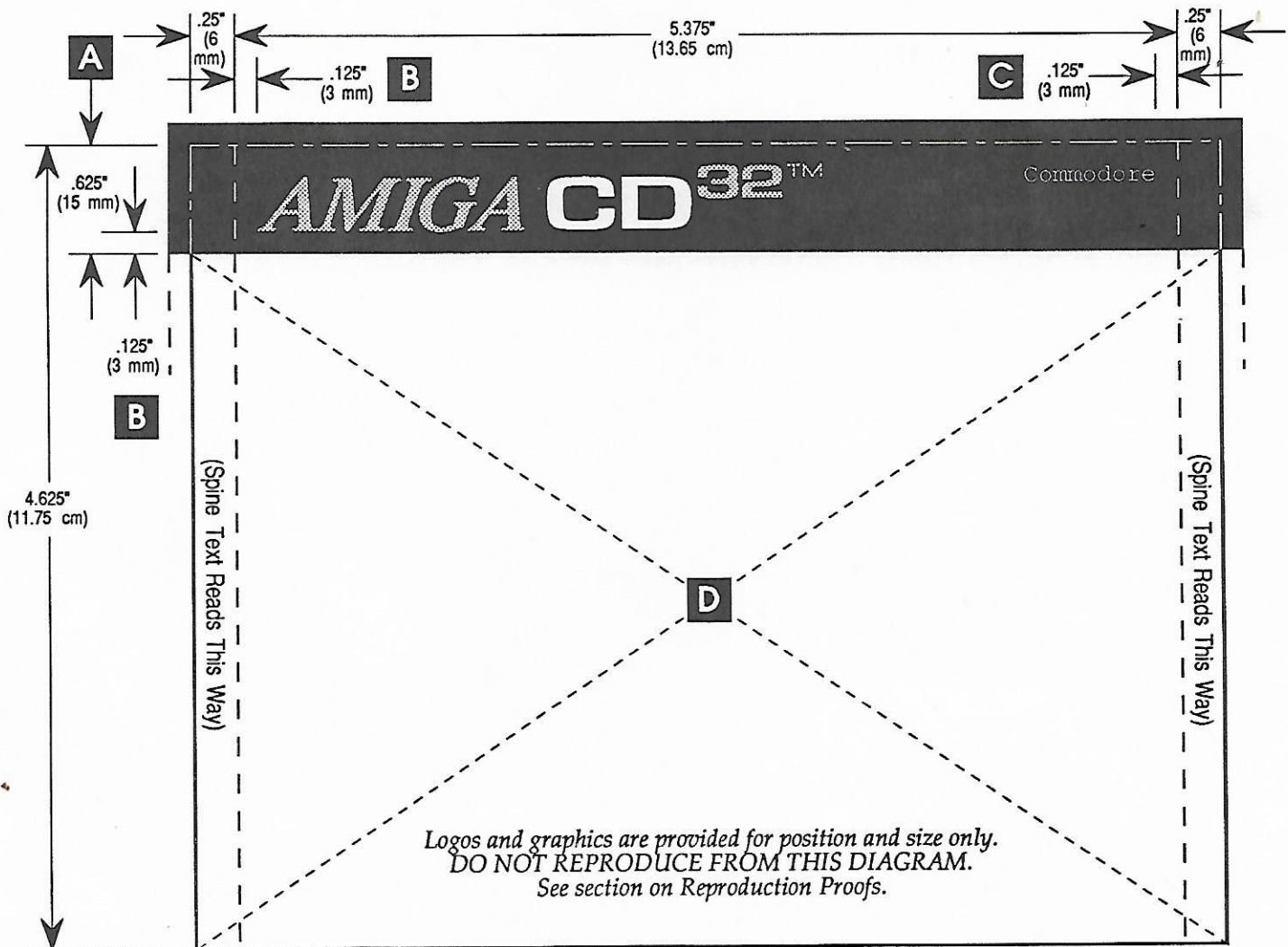


D Inlay Card Graphics

The inlay card is a flat sheet of paper which is scored and folded approximately 1/4" from both the left and right edges. When encased inside a jewel case, it becomes the graphics for the back and side panels. For a double CD jewel case, an identical inlay card is required for front and spine artwork.

Before designing the inlay card and choosing paper stock, check with your disc duplicator for exact measurements and paper requirements. This will avoid jamming or damage during insertion.

- A TOP BAR:** A band of 100% black, 5/8" tall should bleed off the top and both sides of the card.
- B AMIGA CD³² LOGO:** Positioned as shown.
- C COMMODORE LOGO:** positioned so that the top lines up with the top of the Amiga CD³² logo.
- D FOR PUBLISHER USE:** The area within the dashed borders is free to use for title graphics.



Index Card Graphics

The following is a list of the index card graphics that are available in the software. The graphics are arranged in alphabetical order by the first letter of the graphic name. The graphics are arranged in alphabetical order by the first letter of the graphic name.

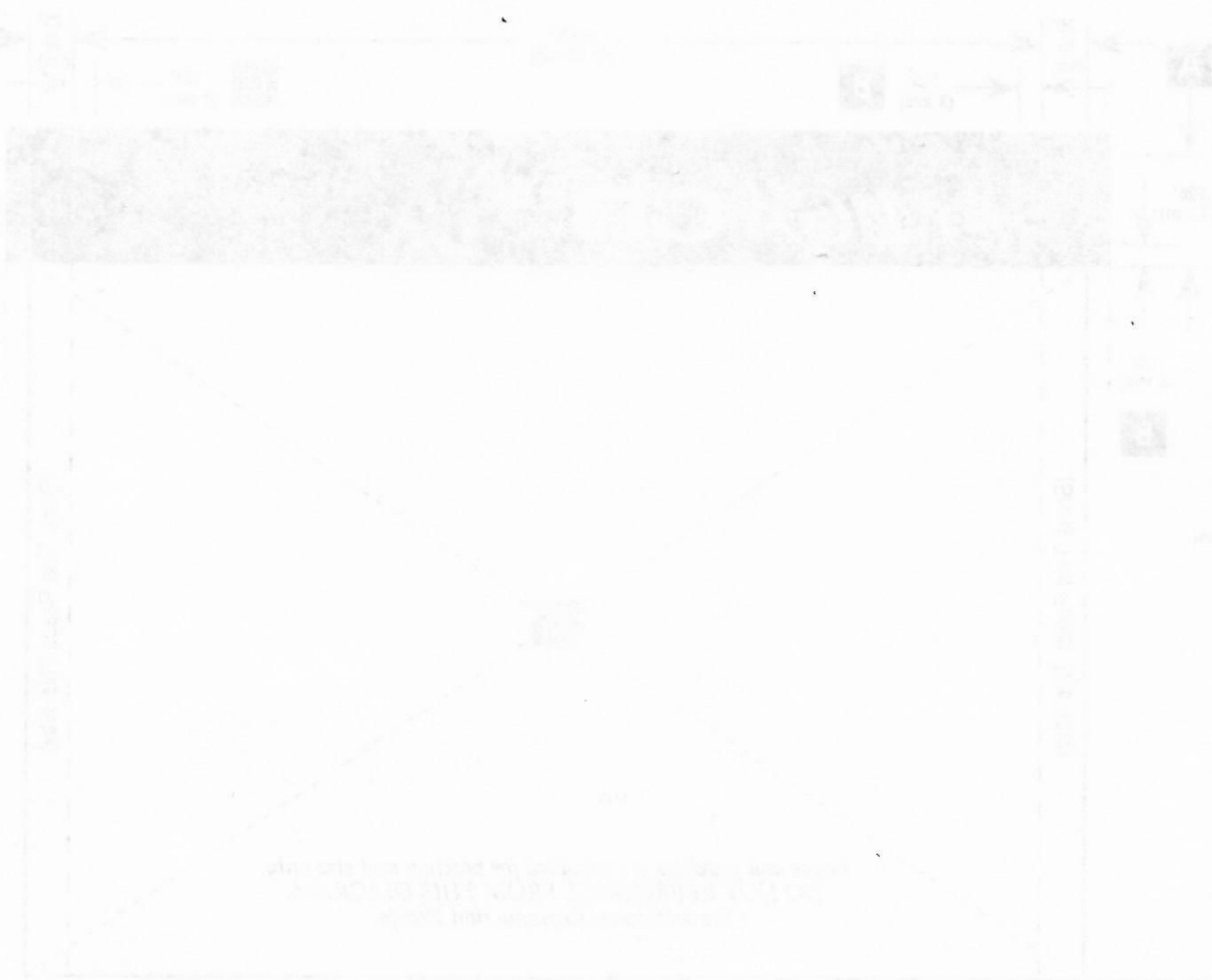
Below is a list of the index card graphics that are available in the software. The graphics are arranged in alphabetical order by the first letter of the graphic name. The graphics are arranged in alphabetical order by the first letter of the graphic name.

Below is a list of the index card graphics that are available in the software. The graphics are arranged in alphabetical order by the first letter of the graphic name. The graphics are arranged in alphabetical order by the first letter of the graphic name.

Below is a list of the index card graphics that are available in the software. The graphics are arranged in alphabetical order by the first letter of the graphic name. The graphics are arranged in alphabetical order by the first letter of the graphic name.

Below is a list of the index card graphics that are available in the software. The graphics are arranged in alphabetical order by the first letter of the graphic name. The graphics are arranged in alphabetical order by the first letter of the graphic name.

Below is a list of the index card graphics that are available in the software. The graphics are arranged in alphabetical order by the first letter of the graphic name. The graphics are arranged in alphabetical order by the first letter of the graphic name.



3

Writing OS-Friendly Games

Many Amiga programmers believe that the only way to write a whiz-bang, speed-of-light game is to bypass the operating system and go straight to the hardware. A better approach is to write games that are OS-friendly. The combination of the AA chipset and the latest V40 system software make system-friendly solutions more possible than ever.

► Why Use the Operating System?

Here are some reasons to use the OS for games:

- OS code runs out of ROM, which is faster than running equivalent code out of Chip RAM.
- Why reinvent the wheel? Spend your time doing things that only you can do.
- The OS automatically supports pre-ECS, ECS, and AA (and will support the next revision of the Amiga chip set).
- There is less code to write. The OS has routines for handling all screen positions and scrolls, mouse movement, etc. This means less development time and less time getting the hardware to work, and more time making the game more playable.
- More robustness. For instance, the OS floppy disk code is far less picky about drive parameters than 99% of custom floppy I/O code.
- Hides bugs and quirks of the chip set. The AA chip set has a few bugs which the OS hides from you.
- Multiple platforms. OS code will run on all Amiga-based machines, whatever their flavor.
- Compatibility with future chipsets. The next revision of the Amiga chip set will not be register-level compatible with AA.

► Things the System Cannot Do

A long-term goal of the Amiga operating system is to support the common display tricks often used in games. There are some things that can't be done though:

- Scrolling individual scan lines of a ViewPort
- Using a copper list to fade a AA color register
- Dynamic updating of user copper lists.

All these are planned to be addressed in future OS releases. The goal is to allow these and other types of operations within normal Intuition screens if possible.

► ECS-to-AA Incompatibilities that the OS Handles

If you are coding for the AA chip set (and you ought to be) the OS handles some problems of backward compatibility with the older ECS chip set that you do not want to deal with.

- Vertical counter behaves differently in programmable beam modes.
- No SuperHires color scrambling.
- Bitplane alignment problems.

► Compatibility Beyond AA that the OS Handles

The successor to the AA chip set will not be register-level compatible. To minimize the side-effects, the operating system will have features designed to hide any incompatibilities from programs that use system calls. Hence, using the operating system for display operations is the best way to ensure a measure of forward-compatibility with chip sets to come.

Some of the problems the system will handle for you are:

- Future chip sets will have no fetch-mode selections. All selections will be automatic.
- Different DDFSTART/STOP calculations for single/dual systems.
- Color loading will be different
- Exact copper timings will be different
- No SuperHires
- Multiple blitters

► Booting A Title

Amiga CD³² boots just like a normal Amiga. The only available boot device is normally the CD (CD0:), so the system boots off the CD in the same manner it would boot from a hard-drive, executing a startup-sequence and so on. In order to boot, your title must have the Commodore trademark file right after the PVD sectors (see chapter 2 for more on this.)

Competing game consoles generally have instantaneous startup times for titles, due to the cartridge interface used. Amiga CD³² benefits in many ways from its CD-ROM technology, however booting time is not one of these benefits. Due to the very slow seek times of CD-ROM devices, the dynamics of disk I/O are quite a bit different from a hard drive. It is crucial that developers be aware of this situation and take appropriate action to reduce boot times.

The startup animation sequence provided by the Amiga CD³² ROM has some continuous motion to indicate that the system is doing something while a title is booting. This animation sequence requires little memory and gets out of the way as soon as the title is ready to run. This animation helps eliminate the long periods of black screen familiar to CDTV users.

An important factor in bootup time is the length of the startup-sequence executed by the system prior to running a title. The startup-sequence shipped with standard Amiga systems is intended for use in a multitasking computer environment. Understanding the startup-sequence, and optimizing it can help reduce bootup time considerably.

The following is a startup-sequence which is adequate for most titles. It is much smaller and many times faster than the standard startup-sequence provided with Workbench 3.1. It also leaves noticeably more memory for your title than when the standard startup-sequence is used.

```
C:SetPatch QUIET

; Only include this line if you are using locale.library
C:Assign >NIL: LOCALE: SYS:Locale

; Here you can add any assignments that are specific to your title.
C:Assign >NIL: WestChesterWarrior: SYS:WestChesterWarrior

; Only include these lines if you include the corresponding monitor files on
; your disc. Only include the files on disc if you actually use the mode these
; monitors provide. Each monitor included eats up memory and takes time to load.
DEVS:Monitors/DblNTSC
DEVS:Monitors/DblPAL

; Here you just start your title
SYS:WestChesterWarrior
```

Note that the use of *any* Workbench files on a CD (other than plain text files such as the startup-sequence) requires a Workbench License Agreement with Commodore. Contact CATS for licensing information.

Using explicit path specifications as shown above (C:Assign instead of just Assign) avoids a lot of head seeking and can improve startup performance. Another important consideration in bootup time is the size and complexity of the title itself. It is highly desirable to get your title on the screen as soon as possible. This can become difficult with very large titles which take a long time to load. The solution to this problem involves either the use of overlays, or the use of a separate loader program.

The simpler approach is the loader program. This is a very small program that gets run in the startup-sequence (just like WestChesterWarrior in the example above). This loader program contains some animation and music sequences that can quickly be put onto the screen. While this animation is playing, the loader program can then proceed to load the rest of the title into memory and activate it. See the DOS RunCommand() or System() functions for information on how the loader program can invoke the main title.

Applications designed to run on other Amiga systems (in addition to Amiga CD³²) might require a more elaborate startup-sequence. In that case be aware that references to DF0: in the startup-sequence will cause a boot failure on an Amiga CD³² system. Similarly, a startup-sequence that executes other scripts must have T: assigned to RAM: in order to work properly on Amiga CD³² systems.

Startup Environment

While the system is loading a title into memory, there is a small animation playing on the screen. This animation tells the user that the system is busy and will be done shortly. This animation consumes little memory, and exits the system as soon as the title indicates it is ready to run. Your application can control this animation with a special Amiga CD³² library, the freeanim.library. Here are some things to keep in mind for dealing successfully with the startup animation:

- The startup animation runs in an Intuition screen. In order for the exit sequence of the animation to work correctly, titles must not take over the display prior to the animation shutting down.

- The startup animation currently does not use the audio.device but this could change in future versions. It is therefore important for titles to avoid playing any music prior to the animation shutting down unless they first ensure that all channels on the audio.device are available (i.e., not in use).
- When the animation exits the system, it leaves the display completely black. You can assume this in your titles, and fade up from black, or whatever other transitions you wish to do.

Once your title is in memory and wishes to display something on the screen, it needs to do:

```
FreeAnimBase = OpenLibrary("freeanim.library", 0);
```

This will tell the startup animation to begin shutting down, which involves removing itself from the display and freeing the resources it uses. This may take a while, up to about a second. While the startup animation is shutting down, your application can prepare data to display and generally initialize itself. This parallelism reduces the time the user has to wait for the titles to start.

Once you have done what you need to in order to display your title screen, you should do the following:

```
CloseLibrary(FreeAnimBase);
```

This will wait for the animation to complete its shutdown. Once this function has returned, it is now safe to display your title screen. If you have no need to process information while the animation is shutting down, the following will work:

```
CloseLibrary(OpenLibrary("freeanim.library", 0));
```

It is important to note that users may try to run your code on a regular Amiga computer equipped with CD-ROM, i.e. not an AmigaCD³². In that case, there will not be a freeanim.library around, so your code should not fail when freeanim.library cannot be opened. This is easy to do, just don't put any error checking in:

```
FreeAnimBase = OpenLibrary("freeanim.library", 0);  
/* initialize application stuff */  
CloseLibrary(FreeAnimBase);  
/* start application animation/display */
```

CloseLibrary() accepts a NULL pointer and ignores it. Therefore if OpenLibrary() fails and returns NULL, CloseLibrary() will still work.

▶ Running Your Titles From Workbench

If Commodore decides to make Amiga CD³²-compatible hardware available for its other Amiga computer systems, it would be highly desirable if any existing Amiga CD³² titles could be run from Workbench on those systems. This requires only a few lines of code to support:

1. You should handle disc insertion and removal events. On a computer system, it would not be wise to enable the "reboot on removal" feature of cd.device. When running on an Amiga CD³², you should generally reboot when a disc is removed, but in a computer environment, you should deal with this eventuality and handle it gracefully by putting up an error message, or at least quitting the title. Crashing is never a good idea.

2. Your code should handle the Workbench startup environment.
3. Your CD should include icons for the appropriate files.
4. Your code should be able to run without having the startup-sequence from your CD run. This is important. If your program relies on assignments made in the startup-sequence, these will not be present when running from Workbench. An easy solution is to provide an IconX script on the CD that gets run from Workbench and loads in the title. The script can do the needed assigns.
5. Your code should multitask nicely. It should release all its resources on exit. If this is not possible, you must warn the user in the documentation and in the software itself that starting this title will take over the system and cause any unsaved work to be lost.

The startup environment in a computer system will be different from the normal Amiga CD³² environment. The `freeanim.library` will not be there, so your code should not fail if `freeanim.library` cannot be opened. In addition, there will not be a startup animation in the system, so the initial display will not be a black screen and will instead be the user's Workbench screen.

It would also be a nice touch to make the software runnable when not on the CD. This would allow users to copy the title to their hard drive, enabling potentially much faster operation.

Finally, if you wish your application to look right at home on a CD-ROM-equipped Amiga computer, you should avoid direct references to "Amiga CD³²" within your title. The title might actually be running on an A1200 or A4000.

■ Preferences

CDs should not contain any preferences files. This specifically means that the following files should not be on your CDs:

```
Devs/system-configuration
Prefs/env-archive/sys/#?
```

Including any preferences file can cause odd behavior during the startup animation sequence.

In addition, many CDTV titles used to call the `Intuition SetPrefs()` function in order to change screen positioning and centering. This is not the correct method and will lead to problems with the startup animation. See the section below on screen positioning for information on how to do it right.

■ Using System Graphics

There are three basic ways you can set up a display on the Amiga:

- Intuition library `OpenScreen()`
- Graphics library `MakeVPort()`, `MrgCop()`, `LoadView()`
- Hitting the hardware registers directly

Using Intuition to set up the display is by far the easiest but also has the greatest memory and processing overhead. A reasonable middle course is to set up the display using the lower-level functions provided in the graphics library. In that case, you do not have to coexist peacefully with Intuition (nor do you get any of the nice services it provides). A detailed example showing how to set up your own display with the `graphics.library` is listed at the end of this chapter.

The worst method of setting up the display is by hitting the hardware registers directly. As the Amiga chip set improves, so many of the internal graphic mechanisms will change that applications using direct access techniques really have no chance of compatibility with future machines.

Graphic Displays

The typical user of the game machine will be using it on the family television set. As such, the display mode that makes the most sense is a low-resolution, 320 x 200 (NTSC) or 256 (PAL), 8-bitplane, 256-color screen. Most games today use the entire display area (i.e., bezel to bezel). To do this on the Amiga requires the use of overscan (more on this in the next section).

The game machine has Chip RAM only, hence, memory is shared between the custom graphics chips and the CPU. Contention between the custom chips and the CPU is an issue to consider. A low-resolution, 8-bitplane display does not involve any cycle stealing from the CPU assuming that the OS has been left in place so that the higher fetch-bandwidth modes of the system are in effect.

Screen Positioning with the OS

With overscan, the Amiga's nominal display dimensions can be extended so that graphics are displayable in what is normally the border area. Prior to Release 2 of the operating system, there was no officially supported way to control overscan displays and only limited support for screen positioning control. Numerous ad hoc techniques of varying quality were invented, and most are still supported, though some only work because of patches designed to keep them operational.

Release 2 (V37) introduced the graphics database and a host of structures to help you control your screen position. For correct operation on the Amiga CD³² console, we strongly urge you to use these functions and structures. First, some definitions to help you along:

Text Overscan: this is the old "morerows" size from 1.3, and has been called the "Text Overscan" or "Text Size" in V37+ Overscan Prefs. This rectangle defaults to a nominal-sized rectangle, but can be enlarged and/or repositioned by the user via Overscan Prefs. If this rectangle is correctly set, no pixel in Text Overscan is supposed to be masked by the bezel of the monitor.

Display Clip: a rectangle that describes the desired displayable portion of your screen. The coordinates are in screen-pixels, and the origin is the upper-left corner of the Text Overscan rectangle.

It is best to center your own display with respect to the Text Overscan rectangle. Here's a piece of code to handle it:

```
/* CenterDClip( displayID, width, height, rect )
 * Fills in the struct Rectangle pointed to by rect with a DisplayClip
 * whose size is the width and height specified, and whose position
 * is centered around the Text Overscan rectangle for the specified
 * displayID.
 *
 * Returns TRUE if it succeeds, FALSE if it doesn't (usually caused
 * by an invalid displayID).
 */

BOOL CenterDClip(ULONG displayID, ULONG width, ULONG height,
                 struct Rectangle *rect )
```



```

{
    BOOL success = FALSE;
    LONG excess_x;
    LONG excess_y;

    if ( QueryOverscan( displayID, rect, OSCAN_TEXT ) )
    {
        excess_x = ( width - ( rect->MaxX - rect->MinX + 1 ) );
        excess_y = ( height - ( rect->MaxY - rect->MinY + 1 ) );
        rect->MinX -= excess_x / 2;
        rect->MaxX = rect->MinX + width - 1;
        rect->MinY -= excess_y / 2;
        rect->MaxY = rect->MinY + height - 1;

        success = TRUE;
    }
    return( success );
}

```

Sample use:

```

#define MY_DISPLAY_ID HIRES_KEY
#define MY_WIDTH      whatever
#define MY_HEIGHT     whatever
...

struct Rectangle dclip;

if ( CenterDClip( MY_DISPLAY_ID, MY_WIDTH, MY_HEIGHT, &dclip ) )
{
    if ( myscreen = OpenScreenTags( NULL,
    SA_DClip, &dclip,
    ...
    TAG_DONE ) )
}

```

Centering yourself using CenterDClip() could theoretically give you a Display Clip which extends outside of the OSCAN_MAX Display Clip, which represents the hardware limits. If you would rather be slightly off-center than clipped, you may wish to shift the resulting DClip to fit within Maximum Overscan. The easiest way to get that rectangle is:

```
QueryOverscan( displayID, &maxrect, OSCAN_MAX );
```

► New V39 and V40 Graphics Capabilities

The graphics.library for V39 gained many features that make creating system friendly, fast moving graphics and animations much easier. Autodocs for the graphics.library are a good source of information on these new features. Some examples are also available from the 1993 DevCon notes. Of specific interest is the new double-buffering support which is even available in Intuition screens.

V40 of graphics.library which is present in the Amiga CD³² ROM contains a few extra features specifically targeted at improving the performance of animation and rendering code. These features are relatively new and not well known, so they are briefly explained below.

ViewPorts that don't load colors. ViewPorts normally always have a copper list associated with them which loads up all the colors needed for their depth. This new feature makes it so that only color 0 gets loaded within the copper list. For deep screens, this will substantially reduce the copper list length. The rest of the colors are inherited from the viewport above. This feature is useful to reduce the inter-Viewport gap, and for faster color cycling. See the VC_NoColorPaletteLoad tag in <graphics/videocontrol.h> and in the graphics.library/VideoControl() Autodoc entry.

Preventing intermediate copper list updating. This speeds up LoadRGB(), SetRGB(), ScrollVPort(), and ChangeVPBitMap(). See the VC_IntermediateCLUpdate tag in <graphics/videocontrol.h> and in the graphics.library/VideoControl() Autodoc entry.

Dual-playfield disable. Disables setting of the dual playfield bit in dual-playfield ViewPorts. Odd and even bit-planes will still scroll separately, but will be fed directly to the palette. This can be used for instance to do cross-fades between independently scrolling images. See the VC_DUALPF_Disable tag in <graphics/videocontrol.h> and the graphics.library/VideoControl() Autodoc entry.

Amiga CD³² includes very fast chunky-to-planar conversion hardware. This allows for high-quality 3D graphics. Consult the documentation for the WriteChunkyPixels() function in the graphics.library Autodoc file for more information. [Note: this function is not yet in the FD or protos but there is a pragma in include/pragmas/graphics_pragmas.h]

Note that the above new ViewPort features work fine in Intuition screens as well. You just need to pass the screen's ViewPort as a parameter to the VideoControl() function.

You might also wish to look at the documentation for the SA_VideoControl and SA_MinimizeISG tags for the OpenScreenTagList() function. These two tags offer Intuition-level control of the new features outlined above. There are also the SA_Interleaved, SA_Exclusive, and SA_Draggable tags which can greatly help writing animation and graphics code. All these tags are documented in <intuition/screens.h> and in the intuition.library Autodoc.

V40 also added many new display modes, four of which are especially useful for animation support. These modes are defined in <graphics/modeid.h> as:

```
/* Added for V40 - may be useful modes for some games or animations. */
#define LORESSDBL_KEY          0x00000008
#define LORESHAMSDBL_KEY      0x00000808
#define LORESEHBSDBL_KEY      0x00000088
#define HIRESHAMSDBL_KEY      0x00008808
```

These new V40 display modes allow you to display a given raster at twice its normal height. Each line of the raster automatically gets repeated on the screen as it is displayed. This allows a 128 pixel high raster to be displayed as if it were 256 pixels tall. This can be very helpful in order to create full screen animations. For example, a 128 pixel high CDXL animation can suddenly occupy the full screen.

When you use one of these new modes, you *do not* allocate more raster or open a taller screen, i.e., you still allocate a raster of the smaller height (128 for example), and if you are blitting the image you still only blit 128 lines. Doubling the height of your image is a video effect accomplished by the AA hardware. These modes work on a composite monitor or TV.

For more on how to set up a display using graphics.library, see the example at the end of this chapter.

► Display Mode Promotion

Mode promotion is a feature which converts 15kHz screens into 31kHz screens, in order to remove interlace flicker, or inter-line gaps. It is the AA equivalent of the A3000 display enhancer hardware.

Mode promotion can be turned on or off by the user. When turned on, screens opened using the 1.3 OpenScreen() function, or opened using a default monitor ID will end up promoted to a 31kHz mode. The promotion process is not totally transparent as certain aspects of the copper list, the refresh rate, and available overscan dimensions are different for the promoted state than for the unpromoted state. These differences can be crucial for many developers.

To avoid promotion affecting your titles, you simply need to open your screens with explicit mode IDs using the `PAL_MONITOR_ID` and `NTSC_MONITOR_ID`, instead of using `DEFAULT_MONITOR_ID`.

Note that promotion is not turned on in Amiga CD³² systems, the problem can only surface in a computer environment.

Consult the ROM Kernel manuals, the 1991 and 1993 DevCon notes for more information on display mode IDs and display mode promotion.

► Intuition Screen Defaults

Applications that use Intuition to set up their displays need to understand how Intuition defaults work. The default values for the `LeftEdge`, `TopEdge`, `Width`, and `Height` of a screen can be a bit complex. This is partially due to the fact that they are implicitly specified by supplying a non-NULL `NewScreen` pointer to `OpenScreenTagList()`, and by the fact that while Intuition does define `STDSCREENWIDTH` and `STDSCREENHEIGHT` magic constants, there are no corresponding constants for standard left-edge and standard top-edge (which are likely non-zero!). The most foolproof strategy is to use a NULL `NewScreen` structure, using only tags to specify screen attributes, and omit all of `SA_Left`, `SA_Top`, `SA_Width`, and `SA_Height`. The correct defaults will come from the `SA_DClip`.

The other way (which is required if using a non-NULL `NewScreen`), is to supply the correct values for each component (either in the appropriate members of the `NewScreen` structure or with the appropriate tags). The correct values are:

```
left    = dclip.MinX
top     = dclip.MinY
width   = dclip.MaxX - dclip.MinX + 1
height  = dclip.MaxY - dclip.MinY + 1
```

Note that if your screen is shorter than the nominal height, other screens that are behind you can potentially be visible above your screen.

You are strongly encouraged to use Intuition screens instead of custom Views and ViewPorts. However, if you go that route, you can install a `DisplayClip` into your ViewPort's `ViewPortExtra` structure.

► Screen Position and Sprites

User preference can shift the Intuition View origin, which in turn can affect the number of sprites available on your screen. If you use sprites other than the pointer sprite, you must take this into consideration, because a change in the upper-left corner of the Text Overscan rectangle shifts the entire coordinate system, so even carefully chosen fixed coordinates may get you into trouble. The upper-left corner of the `OSCAN_MAX` rectangle is the one which is at a fixed position with respect to the hardware fetch cycle (for a given mode). You can base a positional calculation on this if necessary.

There are some Display Clip positions for which the number of possible sprites varies depending on which fetch mode is used. The higher fetch modes have the advantage of reducing Chip RAM bus saturation, but you may prefer a lower bandwidth with more sprites. Graphics will drop the bandwidth to bring allocated sprites into view if `MakeVPort()` runs after you obtain your sprites via `GetExtSprite()`. In an Intuition environment, you should use `RemakeDisplay()` after you get your sprites (not call `MakeVPort()` directly).

■ Avoiding The Workbench Screen

If you use Intuition screens in your titles, it is important to note an Intuition "feature" which can get in the way. Whenever the last screen opened in the system is closed, Intuition automatically opens up the Workbench screen. This can be quite cumbersome in titles which perform transitions between different screens.

Here is a short code sample for SAS/C that defeats this Intuition feature. Simply call `CloseScreenQuiet()` instead of calling `CloseScreen()` and all will be fine:

```
ULONG _asm OpenWorkBenchPatch(void)
{
    return(0);
}

VOID CloseScreenQuiet(struct CDUILib *CDUIBase, struct Screen *screen)
{
    APTR OWB;

    if (screen)
    {
        OWB = SetFunction(IntuitionBase, -0xd2, (APTR)OpenWorkBenchPatch);
        CloseScreen(screen);
        SetFunction(IntuitionBase, -0xd2, OWB);
    }
}
```

Note that the above code will only work starting with V39 Kickstart. Previous Kickstarts did not support this. Amiga CD³² has a V40 Kickstart.

■ Localization

There are two ways software can be localized on Amiga CD³² systems. The `lowlevel.library`, discussed later in this chapter, offers a very simple mechanism that returns the number of the user's current preferred language. The second method involves using `locale.library` which provides a complete suite of localization services.

If your title does not use `locale.library`, it is best to remove the library from your CD. If your title does use `locale.library`, you must have a directory structure such as:

```
Locale (dir)
Catalogs (dir)
    français (dir)
        westchesterwarrior.catalog
    deutsch (dir)
        westchesterwarrior.catalog
    italiano (dir)
        westchesterwarrior.catalog
Languages (dir)
    français.language
    italiano.language
    deutsch.language
```

That is, you need a `Locale` directory which contains a `Catalogs` and a `Languages` subdirectory. There is no need to include the `Countries` directory that comes with the Workbench disk, this directory is only of use to the `Locale` preferences editor.

The Catalogs directory must contain one subdirectory for each language that your title supports. Within these directories, you can put the standard locale.library catalogs files (see locale.doc and catcomp.doc for more information).

The Languages directory must contain a language driver for each of the languages your title supports. These language drivers are supplied with Workbench 3.1 on the Locale disk. The LOCALE: assign must exist if you use locale.library. It should be made to the Locale directory described above.

■ New Amiga CD³² System Software

Five special system modules were created for the AmigaCD³² console:

- lowlevel.library
- nonvolatile.library
- freeanim.library
- cd.device
- CD-ROM file system

You can use these modules with full confidence that they will be upwardly compatible with future systems from Commodore. The lowlevel.library, nonvolatile.library and the CD-ROM file system will be part of Workbench 3.1 and will be shipped within the 3.1 Enhancer kits and with new machines that include Workbench 3.1. The freeanim.library will not be included with the normal Workbench distribution, but your code does not need to rely on its presence (see below for more on this). Finally, cd.device will be included with any Amiga CD³²-compatible solutions that Commodore will offer for its computer systems in future.

■ The nonvolatile.library

The purpose of this library is to allow an application to save small amounts of information to nonvolatile memory or a disk device without having to burden the application with determining which device to use. It provides a simple, common access method to whichever device the user chooses. This library also provides access to true nonvolatile memory present on the game machine. See the nonvolatile.library Autodocs in Chapter 4 for more information on this library.

■ The lowlevel.library

The lowlevel.library provides many functions useful for game developers. Among other things, this library provides the only means to read the Amiga CD³² game controller. In addition to the game controller support code, the lowlevel.library provides many functions often needed by game developers. Most of the functions in this library should not be used in the Amiga's normal multitasking environment—instead they are appropriate only for high-performance games which require the takeover of the system.

■ Compatibility with CDTV

Although the Amiga CD³² console can run old CDTV titles, you should never use any CDTV-specific system modules in your applications. There are known bugs and limitations in these modules when they are used in the Amiga CD³² environment. Future systems will not support these modules:

- cdtv.device
- playerprefs.library
- cardmark.device
- bookmark.device
- rmtm and CDTVPrefs

Game Programming Problems and Solutions

Speed of graphics rendering is the reason programmers usually cite for taking over the system. This is not a very good reason. In this section, some alternatives are described that you should consider instead of directly programming the hardware registers.

Blitter

If you have your own blitter routines that are faster than the routines in ROM, you can use them in a way that is friendly to the operating system. Use `OwnBlitter()` and `DisownBlitter()` to claim and relinquish ownership of the blitter registers.

First call `OwnBlitter()`, do any required setup, call `WaitBlit()`, poke the blitter registers, and then `DisownBlitter()` when all blits are done. This is a very low-overhead way to use direct hardware access; `OwnBlitter()` is only 3 instructions (counting RTS) when no one else is trying to use the blitter. You must use the graphics.library `WaitBlit()`. This is as fast as possible, uses no CPU registers, and knows about blitter bugs. You cannot possibly write one that is more efficient and works on all Amigas.

For asynchronous blits, use the `QBlit()` and `QBSBlit()` functions. These have been rewritten for the 3.0 version of the operating system and now have quite low overhead. These let you use the blitter in an interrupt driven manner instead of polling for completion.

Copper

If you want to do full-screen animation, it is not necessary to take over the system at boot time or directly control the Copper registers. Instead, use a user-Copper list to cause a Copper interrupt on line 0 of your ViewPort.

The copper interrupt handler you install can signal a high-priority task that calls `ChangeVPBitMap()` (or `ScrollVPort()`) and `LoadRGB32()` to change both bitmap pointers and colors in sync. This allows for perfect 60 Hz animation even while moving the mouse as fast as possible and inserting floppy disks.

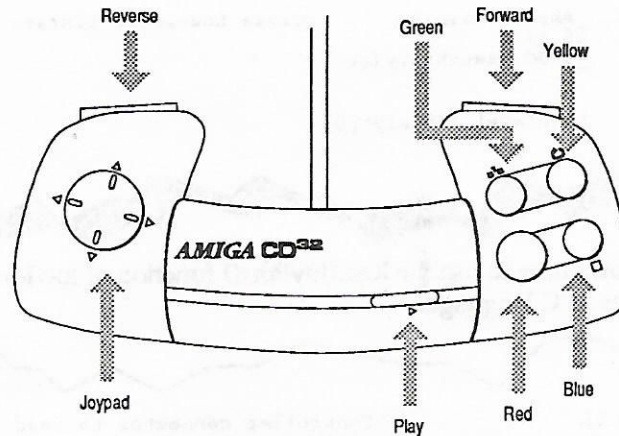
Under 3.0, you can also do this in an Intuition SA_Exclusive screen. You can tell if it was your screen which caused the copper interrupt by checking to see if the flag `VP_HIDE` is clear in your `ViewPort->Modes`.

If you really have to take over the Copper, first call `LoadView(NULL)`, do two `WaitTOF()`s, and then install your own copper lists in the `cop1/2jmp` registers. We do not recommend this though. Future chipsets will have faster and more efficient coppers with different registers. If you load copper registers directly, your code will break on these future systems.

```
temp=GfxBase->ActiView;
LoadView(NULL);
WaitTOF();
WaitTOF(); /* custom.copllc = ??? */
...
WaitTOF();
WaitTOF();
LoadView(temp);
custom.copllc=GfxBase->copinit;
```


Getting User Input

On the game machine, input will usually come from the "game controller." This is a hand-held controller similar to the type used on CDTV and the Nintendo SNES. It plugs into the standard, 9-pin D connectors on the side of the system. It has six action buttons, one start button and four directional arrows. Input could also come from a mouse or joystick.



All three types of game controllers can be conveniently managed with the `ReadJoyPort()` function in the `lowlevel.library`. This function is the only way to get Amiga CD³² game controller information. One drawback of this function is that it requires a polling loop. So applications designed to run in the Amiga's multitasking environment may want to use other methods to get joystick and mouse information. The mouse and joystick information can be obtained from `Intuition`, from the `gameport.device`, or from the hardware port registers directly.

Before looking at the hardware registers directly, it is imperative that you allocate the associated hardware resources using the `GPD_ASKCTYPE` and `GPD_SETCTYPE` commands of the `gameport.device`. You should also allocate the POTGO resource bits to acquire buttons 2 and 3. Code that demonstrates how to do this is listed on pages 89, 94-95 and 344 of the *Amiga ROM Kernel Reference Manual: Devices*, 3rd edition (ISBN 0-201-56775-X).

```

move.l   AbsExecBase,a6
lea      LowLib(pc),a1      ; Open low-level library
move.l   #0,d0              ; version 0
jsr      _LVOpenLibrary(a6)
tst.l    d0                 ; Make sure it opened
beq.w    exit1
move.l   d0,a5              ; Save LowLevelBase in A5

moveq.l  #1,d0              ; Read status of game connector #1
jsr      _LVReadJoyPort(a6) ; with the ReadJoyPort() function.
move.l   d0,d2              ; Save joy 1 status in d2

andi.l   #JP_TYPE_MASK,d0   ; Get the controller type in d0
cmpl.l   #JP_TYPE_MOUSE,d0  ; Is it a mouse?
bne.s    notmouse

;It's a mouse.
notmouse
cmpl.l   #JP_TYPE_JOYSTK,d0 ; It's not a mouse, so...
bne.s    notjoy             ; Is it a joystick?

;It's a joystick

```



```

not joy      cmpi.l  #JP_TYPE_GAMECTLR,d0 ;It's not a mouse or joystick, so...
             bne.s   default             ; Is it an AmigaCD32 game controller?

             ;It's an AmigaCD32 game controller
default      ;It's type is unknown or not available

             move.l  _AbsExecBase,a6      ;Close Low-level Library
             move.l  a5,a1
             jsr     _LVOCloseLibrary(a6)

exit1        rts
lowllib      dc.b    'lowlevel.library',0
             end

```

The fragment above shows how to use the ReadJoyPort() function of lowlevel.library. The same fragment is shown below in C language.

```

#define PORTNO 1L /* Controller connector to read */
struct Library *LowLevelBase; /* Global, define above main() */
...

ULONG status;
GameBase=OpenLibrary("lowlevel.library", 0L );
if(GameBase)
{
    /* Do set up, then, in main loop... */
    status = ReadJoyPort( PORTNO );
    switch( status & JP_TYPE_MASK )
    {
        case JP_TYPE_GAMECTLR: /* It's a game controller */
            break;
        case JP_TYPE_MOUSE: /* It's a mouse */
            break;
        case JP_TYPE_JOYSTK: /* It's a joystick */
            break;
        default: /* Contoller type unknown or unavailable */
    }
    CloseLibrary(LowLevelBase);
}

```

Dealing with the Input Device

On the Amiga, the input.device is the system module that processes input events such as mouse movement and keystrokes passing them on to Intuition and other interested clients. If the system is not being completely shut down by use of the SCON_TakeOverSys tag to SystemControlA() the system overhead can be reduced by shutting down input.device. Obviously this should not be done if your application depends on input.device for gathering input.

There are two ways to do this. With lowlevel.library, the SystemControl() function can be used with the SCON_StopInput tag set to TRUE. This stops the input.device from using any CPU time and also prevents it from passing along input events to Intuition. This is the easiest way to shut down the input.device on the CD game machine.

A solution which is more multitasking-friendly is to install a high priority input handler which chokes off all events. To do this, you use the IND_ADDHANDLER and IND_REMHANDLER commands of

the input.device. Examples of how to use these are listed on pages 108 and 109 of the *Amiga ROM Kernel Reference Manual: Devices*, 3rd edition (ISBN 0-201-56775-X). By using a priority of 51, you can make your input handler first in the chain and prevent other handlers (like Intuition) from seeing input activity. This handler is also a convenient way to get keys and mouse events yourself. Simply store the raw keypresses and mouse moves in your own variables.

The CD game machine in its base configuration does not include a keyboard. However, a keyboard is planned as part of an expansion option for the system. Programs that want to support this option can get keyboard events in a variety of ways. The lowlevel.library functions AddKBInt() and RemKBInt() will add and remove code you provide to the system's keyboard interrupt servicing allowing you to detect raw keypresses.

If you prefer to poll the keyboard as part of your main processing loop, use the lowlevel.library function GetKey(). This returns a raw key value for the current key pressed and also any qualifier keys (such as Shift) that are pressed at the same time. To monitor the state of a given set of keys, you can use the lowlevel.library function QueryKeys(). You pass this function an array of raw key codes for the keys you want to know about and it returns their current state.

Another alternative that is multitasking friendly is to obtain input from the keyboard.device. This has the advantage of dealing with all the various keyboard timings, but it is easier by far to open an Intuition window and read either RAWKEY or VANILLAKEY IDCMP messages. These either give the raw key number pressed, or the character the key pressed represents (useful for international games).

Timing

For calculating elapsed time, there are a variety of possible solutions. For a variable-speed, high-precision, interrupt-driven time source, programs can use the timer functions of the lowlevel.library. These functions provide a way for your code to be called continuously at arbitrary intervals. For instance the example below shows how to wait for precisely 1000 microseconds.

```

struct Library *LowLevelBase;

struct TIData
{
    struct Task *atask;
    LONG        asignal;
};

VOID __asm __interrupt __saved
TIRoutine (register __a1 struct TIData *tidata)
{
    Signal( tidata->atask, 1L << tidata->asignal );
}

VOID
main ( int argc, char **argv)
{
    APTR          ti_handle;
    struct TIData ti_data;

    LowLevelBase = OpenLibrary ( "lowlevel.library", 0L);
    if(LowLevelBase)
    {
        ti_data.atask = FindTask( NULL );
        ti_data.asignal = AllocSignal(-1L);
        if(ti_data.asignal!=-1)
    }
}
/* This is just a structure */
/* to hold parameters passed */
/* to the interrupt routine. */

/* Here's the timer interrupt */
/* routine. It simply signals */
/* the parent task pointed to */
/* by register A1. */

/* lowlevel.library timer handle */
/* Pointer to interrupt arg.s */

/* Fill in TIData structure */
/* so interrupt routine can */
/* signal us. */

```

```

{
/* Attach the TIRoutine to the timer generated interrupt. */
ti_handle = AddTimerInt(TIRoutine, &ti_data);
if(ti_handle)
{
/* Start the timer; TIRoutine will be called every 1000 microsecs */
StartTimerInt(ti_handle, 1000L, TRUE);

Wait(1L << ti_data.asignal);      /* Wait for interrupt signal. */

StopTimerInt(ti_handle);          /* Tidy up. Free any system */
RemTimerInt(ti_handle);          /* resources used. */
}
FreeSignal(ti_data.asignal);
}
CloseLibrary(LowLevelBase);
}

```

Here is a similar example in 68020 assembly language. This time, the example flashes the display every 1024 ticks for a total delay of just over 10 seconds.

```

STRUCTURE TIData,0
LONG asignal
APTR atask
LABEL TIData_sizeof
;

move.l AbsExecBase,a6
lea.l LowLib(pc),a1          ; Open low-level library
move.l #0,d0                 ; version 0
jsr _LVOOpenLibrary(a6)
tst.l d0                     ; Make sure it opened
beq.w exit3
move.l d0,LowLevelBase       ; Save the base so we can close later

lea.l IntuLib(pc),a1          ; Open Intuition Library
move.l #0,d0                 ; version 0
jsr _LVOOpenLibrary(a6)
tst.l d0                     ; Make sure it opened
beq.w exit2
move.l d0,IntuitionBase      ; Save the base so we can close later

move.l #0,a1                 ; Call Exec FindTask(NULL);
jsr _LVFindTask(a6)           ; to get a pointer to the process.
lea.l ti_data(pc),a0
move.l d0,atask(a0)          ;

move.l #-1,d0                 ; Allocate a signal with Exec
jsr _LVOAllocSignal(a6)       ; AllocSignal(-1L)
cmpi.l #-1,d0
beq.w exit1
lea.l ti_data(pc),a0          ; Store the allocated signal number
move.l d0,asignal(a0)        ; in the asignal field of ti_data

move.l LowLevelBase(pc),a6    ; Add in the timer interrupt
lea.l TIRoutine(pc),a0
lea.l ti_data(pc),a1
jsr _LVOAddTimerInt(a6)
tst.l d0
beq.w exit0
move.l d0,ti_handle

move.l d0,a1                 ; Start the timer
move.l #1000,d0
move.l #TRUE,d1
jsr _LVStartTimerInt(a6)

move.l #10*1024,d4           ; Register d4 is the loop counter

```



```

mainloop
    move.l    _AbsExecBase,a6
    lea.l     ti_data(pc),a0
    move.l    asignal(a0),d1
    move.l    #1,d0
    asl.l     d1,d0
    jsr       _LVOWait(a6)

    move.l    d4,d0
    andi.l    #$000003ff,d0          ;Flash the screen every 1024 ticks
    bne.s     looptest

    move.l    IntuitionBase(pc),a6
    move.l    #0,a0
    jsr       _LVODisplayBeep(a6)

looptest
    subi.l    #1,d4
    tst.l     d4
    bne.w     mainloop

    move.l    LowLevelBase(pc),a6
    lea.l     ti_handle(pc),a1
    jsr       _LVOSTopTimerInt(a6)

    move.l    ti_handle(pc),a1
    jsr       _LVORemTimerInt(a6)

exit0
    move.l    _AbsExecBase,a6          ;Free the signal stored in the asignal
    lea.l     ti_data(pc),a1          ;field of ti_data
    move.l    asignal(a1),d0
    jsr       _LVOfreeSignal(a6)

exit1
    move.l    IntuitionBase(pc),a1      ;Close Intuition library
    jsr       _LVOCloseLibrary(a6)

exit2
    move.l    LowLevelBase(pc),a1      ;Close low-level library
    jsr       _LVOCloseLibrary(a6)

exit3
    rts

TIRoutine
    move.l    _AbsExecBase,a6          ;Prepare to call Exec Signal()
    move.l    asignal(a1),d1          ; register d0 contains the
    move.l    #1,d0                   ; signal mask created from asignal
    asl.l     d1,d0                   ;
    move.l    atask(a1),a1            ; register a1 points to the task
    jsr       _LVOSignal(a6)
    rts

;===== Data Area
LowLevelBase    dc.l    0
IntuitionBase   dc.l    0
loopcount       dc.l    0
ti_handle       dc.l    0
ti_data         ds.b     TIData_sizeof,0
                cnop 2
lowllib         dc.b     'lowlevel.library',0
                cnop 2
intulib         dc.b     'intuition.library',0
end

```

The lowlevel.library timer functions use the CIA chips (or equivalents) as the source for timing information. On systems prior to V40 that have no lowlevel.library, you will have to use the CIA timing facilities at a lower level. This involves a lot more housekeeping on the part of your application. For example code demonstrating how to do this, refer to pages 328-335 of the *Amiga ROM Kernel Reference Manual: Devices*, 3rd edition (ISBN 0-201-56775-X).

Another timing source is available through the timer.device called the E-Clock. The E-Clock is the clock used by the Motorola 680x0-family of processors to communicate with Motorola 8-bit devices. ReadEClock() is a timer.device function that provides a simple, low overhead way to determine

elapsed E-Clock time. It returns two values: a 64-bit timer value (the E-Clock count) and a 32-bit frequency value in ticks/second (the E-Clock frequency, related to the master frequency of the machine and different between PAL and NTSC systems). Usage of the E-Clock is shown on page 298 of the *Amiga ROM Kernel Reference Manual: Devices*, 3rd edition, (ISBN 0-201-56775-X). There is also a convenient interface to the EClock provided by the lowlevel.library function `ElapsedTime()`. See the Autodocs in Chapter 4 for more information.

For timings that require less granularity, the system vertical blank can be used as a special low-frequency timer. `WaitTOF()` will suspend your task until the top of frame is reached (and all vertical blank interrupt servers have been called). The frame rate depends on whether the system is running in a PAL or NTSC environment but is totally independent of the processor speed:

```
ULONG frate;
frate = (GfxBase->DisplayFlags & PAL) ? 50 : 60;
for( i=0; i < frate; i++ )
    WaitTOF();
```

As a convenience, V40 of the operating system includes a new field in `GfxBase` named `VBCounter`. The `VBCounter` field is a longword which is incremented by the system every vertical blank.

Audio

The audio device is one of the simplest, lowest overhead modules in the system. In general, there is no reason to hit the hardware directly. There are device commands to change the volume, period, frequency and data that is played on any of the four channels.

Some applications may want to modulate one voice with another or exceed the 28,000 byte/second sample playback rate by using CPU-driven audio instead of DMA-driven audio. In that case, direct

```
VOID      main( int argc, char **argv)
{
    struct IOAudio *myAIReq=NULL;
    struct MsgPort *myAIReply=NULL;
    UBYTE chans[] = {15};          /* Allocate all 4 audio channels */
    BYTE dev = -1;

    myAIReq=(struct IOAudio *)AllocMem(sizeof(struct IOAudio),MEMF_PUBLIC );
    if(myAIReq)
    {
        myAIReply=CreatePort(0,0);
        if(myAIReply)
        {
            myAIReq->ioa_Request.io_Message.mn_ReplyPort = myAIReply;
            myAIReq->ioa_Request.io_Message.mn_Node.ln_Pri = 127;
            myAIReq->ioa_Request.io_Command = ADCMD_ALLOCATE;
            myAIReq->ioa_AllocKey = 0;
            myAIReq->ioa_Data = chans;
            myAIReq->ioa_Length = sizeof(chans);

            dev=OpenDevice("audio.device",0L,(struct IORequest *)myAIReq,0L);
            if(! dev)
            {
                /* Your code that uses audio hardware registers goes here */

                CloseDevice(myAIReq);
            }
            else { /* Couldn't get all 4 channels. */ }
            DeletePort( myAIReply );
        }
        FreeMem( myAIReq, sizeof(struct IOAudio) );
    }
}
```



```

;Imports (amiga.lib) =====
xref _CreatePort
xref _DeletePort
;Exports=====
xdef _SysBase
;Equates=====
SysBase EQU 4
SIZEOFCHANS EQU 1

;Number of bytes in chans, the
; channel allocation array

move.l SysBase,a6
move.l #ioa_SIZEOF,d0
move.l #MEMF_PUBLIC,d1
jsr _LVOAllocMem(a6)
move.l d0,a3
tst.l d0
beq exit3
;Allocate memory for the IOAudio

;Call the amiga.lib C function
; CreatePort() to create a
; reply port
move.l #0,-(sp)
move.l #0,-(sp)
jsr _CreatePort
addq.l #8,sp
move.l d0,a4
tst.l d0
beq exit2
;a3 = a pointer to my IOAudio
; request block

;Initialize the IOAudio request
;myAIOreq
move.w #0,ioa_AllocKey(a3) ; ->ioa_AllocKey
lea.l chans(pc),a1
move.l a1,ioa_Data(a3) ; ->ioa_Data
move.l #SIZEOFCHANS,ioa_Length(a3) ; ->ioa_Length
move.w #ADCMD_ALLOCATE,_IO_COMMAND(a3) ; ->ioa_Request.io_Command
move.l a4,MN_REPLYPORT(a3) ; ->ioa_Request.io_Message.
; mn_ReplyPort
move.b #127,LN_PRI(a3) ; ->ioa_Request.io_Message.
; mn_Node.ln_Pri

lea.l adname(pc),a0
moveq.l #0,d0
move.l a3,a1
moveq.l #0,d1
jsr _LVOpenDevice(a6)
tst.l d0
bne.w exit1
; OK now call open device, unit 0,
; on the audio.device
; Make sure it opened

move.l #SIGBREAKF_CTRL_C,d0
jsr _LVOWait(a6) ;Wait() for Ctrl-C

move.l a3,a1
jsr _LVOCloseDevice(a6) ;Call CloseDevice() to free
; channels
exit1
move.l a4,-(sp)
jsr _DeletePort
addq.l #4,sp
;Delete reply port
exit2
move.l a3,a1
move.l #ioa_SIZEOF,d0
jsr _LVOfreeMem(a6) ;Free the IOAudio memory
exit3
rts

; Data Area =====
chans dc.b 15
cnop 2
adname dc.b 'audio.device',0
end

```

hardware access is required. If you must hit the audio hardware, you can ask for the channel you need with the highest priority (127). The code above shows how:

The audio channels will never be stolen from you, nor will they be used by anyone else until you give the channels back to the system.

Generating Random Numbers

In all games you will need at least some element of randomness in the game to prevent the user from feeling bored during repeat plays. There are a variety of ways to generate random numbers.

The code below is based on the random-number routine from *Semi-Numerical Algorithms: The Art of Computer Programming, Vol. 2*, by Donald Knuth. The function `rand_word()` will return a random 16-bit value and has an extremely long period. The `rand_word()` function uses a table for speed; the table is initialized with another random number generator (which is slightly slower because it has a multiply in it).

```
init table:
; initialize random # generator - pass d0 = a random seed (based off of time)
; trashes d0/d1, a0
    moveq    #54, d1
    lea      rand_table(a6), a0
1$:    move   d0, (a0)+
    mulu     #$1efd, d0
    add      #$dfff, d0
    dbra     d1, 1$
    move.l   a0, (a0)
    move.l   #rand_table+24*2, j_index(a6)
    rts

rand word:
; return a random word in d0
; trashes a0/a1
    lea      rand_table(a6), a1
    move.l   j_index(a6), a0
    move.w   -(a0), d0
    cmp.l    a0, a1
    bne.s    2$
    lea      rand_table+55*2(a6), a0
2$:    move.l   a0, j_index(a6)
    move.l   k_index(a6), a0
    add      -(a0), d0
    move     d0, (a0)
    cmp.l    a0, a1
    bne.s    3$
    lea      rand_table+55*2(a6), a0
3$:    move.l   a0, k_index(a6)
    rts

section variables

;!! rand table must precede k_index
rand_table    ds.w    55
k_index       dc.l    0
;!! k_index must follow rand_table

j_index       dc.l    0

end
```

You will want to use a different seed value each time your program is run. The best way to do this is by using the `ReadEClock()` routine of the timer device to obtain a seed value. The `EClock` is a high speed, free-running, 64-bit timer that increments every 279.5 ns. This is fast enough so that no matter how the program is started, you are very likely to get a different reading from the `EClock` from one game to the next. (See the earlier section on Timing for more information.)

An alternative to the table-driven random number generator above is listed in the `Random()` and `RandomSeed()` assembly-language functions below. These functions are callable from either C or

assembly language. The `amiga.lib` link library provided by Commodore has additional random number routines `FastRand()` and `RangeRand()` that can be called only from a C-language program. Here's a C program that demonstrates how to call these functions to obtain a random number.

```

/* test_random.c - Execute me to compile me with SAS C 5.10
LC -bl -cfist -v -j73 test_random.c
Blink FROM LIB:c.o,test_random.o,random.o TO test_random LIBRARY
LIB:LC.lib,LIB:Amiga.lib
quit

* Shows the use of both the Amiga.lib RangeRand() function and the
* random.asm Random() function it is linked with to demonstrate
* two ways for generating random numbers.
*
* Uses Intuition CurrentTime() to get a seed for the random number generator
* Alternately, you could use the 2.0 timer.device function ReadEClock
*/

#include <exec/types.h>
#include <exec/memory.h>
#include <dos/dos.h>
#include <intuition/intuition.h>

#include <clib/exec_protos.h>
#include <clib/dos_protos.h>
#include <clib/intuition_protos.h>
#include <clib/graphics_protos.h>

#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#ifdef LATTICE
int CXBRK(void) { return(0); } /* Disable Lattice CTRL/C handling */
void chkabort(void) { return; } /* really */
#endif

#define MINARGS 1

UBYTE *vers = "\0$VER: test_random 39.1";

/* Option 1: Random.asm RandomSeed() and Random() and functions
*
* RandomSeed() - Seed with a variable number such as the current time
*
* Random() - Pass limit of 0-65535, returns a number between 0 and (limit-1)
*/
extern void RandomSeed(ULONG seedvalue);
extern int Random(LONG limit);

/* Option 2:
* Amiga.lib ULONG RangeSeed and RangeRand() function
*
* RangeRand() - Pass limit of 0-65535, returns a number between 0 and (limit-1)
*/
#include <clib/alib_protos.h>
extern ULONG far RangeSeed;

void testrand(LONG limit);

struct Library *IntuitionBase;

void main(int argc, char **argv)
{
    ULONG secs,mics,seed;

    if(IntuitionBase = OpenLibrary("intuition.library",0))
    {
        /* We'll use CurrentTime() to get an initial "seed" value.
        * You could also use the 2.0 timer.device function ReadEClock.
        * If you use the same seed value, you'll get the same
        * sequence of "random" values each time you run the program.
        */

```

```

    CurrentTime(&secs,&mics); /* Use as our seed */
    seed = secs + mics;
    printf("secs=%ld mics=%ld, seed = $08lx\n",secs, mics, seed);

    /* We'll be testing both the random.asm Random() function and the
     * amiga.lib RangeRand() function so we need to seed each with a
     * starter random value by using the seed method each requires
     */
    RandomSeed(seed);          /* seed random.asm Random() */
    RangeSeed = seed;         /* seed Amiga.lib RangeRand() */

    /* Call both functions to get some random numbers, passing limit
     * Both functions return random numbers from 0 through limit-1
     * Note however that 0 cannot be passed as limit to RangeRand()
     */
    testrand(3);
    testrand(11);
    testrand(512);
    testrand(65535);

    CloseLibrary(IntuitionBase);
}

void testrand(LONG limit)
{
    int k, ktries;

    ktries = 10;

    /* Note: Can pass limit 0-65535, returns number from 0 to limit-1 */
    printf("Random(): %ld random values between 0 and %ld:\n",ktries,limit-1);
    for(k=0; k<ktries; k++)    printf("%ld ",Random(limit));
    printf("\n");

    /* Note: Can pass limit 1-65535, returns number from 0 to limit-1 */
    printf("RangeRand(): %ld random values between 0 and %ld:\n",ktries,limit-1);
    for(k=0; k<ktries; k++)    printf("%ld ",RangeRand(limit));
    printf("\n\n");
}

```

* Random number generator

XDEF XDEF CODE	RandomSeed,Random _RandomSeed,_Random	assembler entry points C entry points
----------------------	--	--

```

;=====
; NAME
; RandomSeed - seed random number generator, call once at beginning of
;               your program. CurrentTime provides useful values for this
;
; SYNOPSIS
; RandomSeed( SeedValue )
;               DO
;
; FUNCTION
; Seeds the random number generator
;
; INPUTS
; SeedValue - a longword containing any value you like
;
; RESULT
; Random number generator is initialised
;
; BUGS/LIMITATIONS
; None that I know of
;=====

```



```

RandomSeed    MOVE.L 4(SP),D0      entry point for C functions
RandomSeed    ADD.L D0,D1          user seed in d0 (d1 too)
RandomSeed    MOVEM.L D0/D1,RND

; drops through to the main random function (not user callable)

LongRnd       MOVEM.L D2-D3,-(SP)
LongRnd       MOVEM.L RND,D0/D1    D0=LSB's, D1=MSB's of random number
LongRnd       ANDI.B #$0E,D0       ensure upper 59 bits are an...
LongRnd       ORI.B  #$20,D0       ...odd binary number
LongRnd       MOVE.L D0,D2
LongRnd       MOVE.L D1,D3
LongRnd       ADD.L D2,D2           accounts for 1 of 17 left shifts
LongRnd       ADDX.L D3,D3         [D2/D3] = RND*2
LongRnd       ADD.L D2,D0
LongRnd       ADDX.L D3,D1         [D0/D1] = RND*3
LongRnd       SWAP D3             shift [D2/D3] additional 16 times
LongRnd       SWAP D2
LongRnd       MOVE.W D2,D3
LongRnd       CLR.W D2
LongRnd       ADD.L D2,D0          add to [D0/D1]
LongRnd       ADDX.L D3,D1
LongRnd       MOVEM.L D0/D1,RND    save for next time through
LongRnd       MOVE.L D1,D0         most random part to D0
LongRnd       MOVEM.L (SP)+,D2-D3
LongRnd       RTS

;=====
; NAME
;   Random - returns a random integer in the specified range
;
; SYNOPSIS
;   RndNum = Random( UpperLimit )
;   D0
;
; FUNCTION
;   returns a random integer in the range 0 to UpperLimit-1
;
; INPUTS
;   UpperLimit - a long(or short will do) in the range 1-65535
;
; RESULT
;   a random integer is returned to you, real quick!
;
; BUGS/LIMITATIONS
;   range was limited to 0-65535 to avoid problems with the DIVU instruction
;   which can return real wierd values if the result is larger than 16 bits.
;=====

Random        MOVE.L 4(SP),D0      C entry point
Random        MOVE.W D2,-(SP)
Random        MOVE.W D0,D2         save upper limit
Random        BEQ.S 10$            range of 0 returns 0 always
Random        BSR.S LongRnd        get a longword random number
Random        CLR.W D0             use upper word (it's most random)
Random        SWAP D0
Random        DIVU.W D2,D0         divide by range...
Random        CLR.W D0             ...and use remainder for the value
Random        SWAP D0             result in D0.W
10$           MOVE.W (SP)+,D2
10$           RTS

RND           BSS
RND           DS.L 2              random number
RND           END

```

Double Speed and Other CD Drive Issues

The Amiga CD³² has an added feature that allows the CD-ROM drive to perform in a double-speed mode of operation (300Kb/s, 150 frames/s). To have the drive operate in double-speed mode, you need to do two things. The first thing is very important. You need to put your data as close to the outermost tracks as possible. Double-speed works best when your data is on the outer tracks. The ISOCD utility program on the ISO Tools Disk gives you the ability to pad the beginning of the disk allowing you to place your data as far out as possible. The drive will not prohibit you from using double-speed on the inner tracks of the disk, but you should use double-speed only on the outer tracks for best results.

The second thing you need to do is tell the drive to do double-speed. This is done by issuing the CD_CONFIG command of the cd.device as shown in the example below. Depending on the tags you use with the CD_CONFIG command, you can double the speed of either normal reads or CDXL-style reads. Usually, you will only want to configure normal reads for double-speed.

The MaxReadSpeed() function below shows how to configure normal reads (cd.device/CD_READ) to the maximum speed the drive is capable of. The SetReadXLSpeed() below shows how to configure CDXL-style reads (cd.device/CD_READXL) to a desired speed. If the only thing you want to use double-speed for is a CDXL sequence, then you only need to use the SetReadXLSpeed() function. You do not need to configure the CD_READ command to do double-speed as well. Just make sure that you XL sequence is on the outer tracks of the disk.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include <exec/io.h>
#include <exec/memory.h>
#include <utility/tagitem.h>
#include <devices/cd.h>

#include <clib/exec_protos.h>
#include <pragmas/exec_pragmas.h>

extern struct SysBase *SysBase;

struct IOStdReq *IOR;
struct MsgPort *Port;

struct CDInfo CDInfo;
struct TagItem ConfigList[] =
{ 0, 0 },
{ TAG_END, 0 }
};

/*****
 * int MaxReadSpeed(void) -- Set the speed of the READ command to the
 *                          maximum that the drive can handle.
 *****/

int MaxReadSpeed(void) {
    int success = 0;

    if (Port = CreateMsgPort()) {
        if (IOR = CreateIORequest(Port, sizeof(struct IOStdReq))) {
            if (!OpenDevice("cd.device", 0L, IOR, 0L)) {
                IOR->io_Command = CD_INFO;
                IOR->io_Data = (APTR)&CDInfo;
                IOR->io_Offset = 0;
                IOR->io_Length = sizeof(struct CDInfo);
                DoIO(IOR);
            }
        }
    }
}
```



```

        if (!IOR->io_Error) { /* Success? */

            /* Modify TagList */
            ConfigList[0].ti_Tag = TAGCD_READSPEED;
            ConfigList[0].ti_Data = CDInfo.MaxSpeed;

            /* Configure drive to maximum speed */
            IOR->io_Command = CD_CONFIG;
            IOR->io_Data = (APTR)&ConfigList;
            IOR->io_Length = 0;
            DoIO(IOR);

            /* Report success */
            if (!IOR->io_Error) success = 1;
        }
        CloseDevice(IOR); /* Close cd.device */
        DeleteIORequest(IOR); /* Delete IORequest */
        DeleteMsgPort(Port); /* Delete reply port */
    }
    return(success); /* Return success */
}

/*****
 * int SetReadSpeed(int Speed) -- Set the speed of the READXL command to
 *                               the desired frame rate.
 *****/

int SetReadXLSpeed(int Speed) {
    int success = 0;

    /* Create reply port */
    if (Port = CreateMsgPort()) {
        /* Create IORequest */
        if (IOR = CreateIORequest(Port, sizeof(struct IOStdReq))) {
            if (!OpenDevice("cd.device", 0L, IOR, 0L)) { /* Open cd.device */

                /* Modify TagList */
                ConfigList[0].ti_Tag = TAGCD_READXLSPEED;
                ConfigList[0].ti_Data = Speed;

                /* Configure drive to desired speed */
                IOR->io_Command = CD_CONFIG;
                IOR->io_Data = (APTR)&ConfigList;
                IOR->io_Length = 0;
                DoIO(IOR);

                if (!IOR->io_Error) success = 1; /* Report success */

                CloseDevice(IOR); /* Close cd.device */
                DeleteIORequest(IOR); /* Delete IORequest */
                DeleteMsgPort(Port); /* Delete reply port */
            }
            return(success); /* Return success */
        }
    }

/*****
 * Try setting the READ speed to the maximum the drive can handle and try
 * setting the speed of the READXL command to 300Kb/s.
 *****/

void main (int argc, char **argv) {

    /* Set the READ commands to maximum speed */
    if (MaxReadSpeed()) printf("Read commands are at maximum speed");

    /* Try to set READXL frame rate to 150 frames/second (300Kb/s) */
    if (!SetReadXLSpeed(150)) printf("Drive can't support double-speed");
}

```


Other CD Drive Issues

The unit name for the CD-ROM drive on the game machine is CD0:. This is the same name used on CDTV and on the A570 CD-ROM peripheral for the A500. However keep in mind that other vendors who provide CD-ROM drives for the Amiga may use a different drive name. For future compatibility try to use paths relative to your application's volume name rather than relative to the unit name. The system also provides a special process-relative assign named PROGDIR:. This assignment exists for every process except those on the resident list and will be the directory from which the process's executable was loaded.

Another drive consideration is that titles that could run on an A4000 or A1200 equipped with a CD-ROM drive will not be able to boot from CD on such systems (no ISO filesystem in ROM in V39 and earlier). This is one excellent reason to enable your product to run from the Workbench.

Similarly, products that use the CD-ROM at the lowest level, i.e., by making calls to cd.device, are unlikely to work with CD-ROM drives from other vendors whose implementations of the ISO filesystem is different. To avoid this compatibility pitfall, stick to the higher-level filesystem interface to the CD drive. Just use the dos.library.

Memory

The CD game machine will have two megabytes of Chip RAM in its base configuration. There are a fairly large number of wait-states when accessing Chip RAM. ROM is zero wait-states. Due to the slow RAM speed, it may be better to use calculations for some things that you might have used tables for on the A500. Add-on RAM (Fast RAM) will probably be faster than Chip RAM, so it is worth segmenting your game so that parts of it can go into Fast RAM if available.

CPU Speed Issues

With the 68EC020 processor used in all CD game machines, there are some special coding techniques you can use to improve performance. For example, it is critical that you code any important loops to execute entirely from the on-chip 256-byte cache. A straight line loop 258 bytes long will execute far slower than a 254 byte one. The '020 is a 32-bit chip. Longword accesses will be twice as fast when they are aligned on a longword boundary. Aligning the entry points of routines on 32-bit boundaries can help, also. You should also make sure that the stack is always longword aligned. Write accesses to Chip RAM incur wait-states. However, other processor instructions can execute while results are being written to memory:

```
move.l d0, (a0)+ ; store x coordinate
move.l d1, (a0)+ ; store y coordinate
add.l d2, d0 ; x+=deltax
add.l d3, d1 ; y+=deltay
```

will be slower than:

```
move.l d0, (a0)+ ; store x coordinate
add.l d2, d0 ; x+=deltax
move.l d1, (a0)+ ; store y coordinate
add.l d3, d1 ; y+=deltay
```

The 68020 adds a number of enhancements to the 68000 architecture, including new addressing modes and instructions. Some of these are unconditional speedups, while others only sometimes help.

■ New Addressing Modes

Scaled Indexing. The 68000 addressing mode (disp,An,Dn) can have a scale factor of 2, 4 or 8 applied to the data register on the 68020. This is totally free in terms of instruction length and execution time. For example:

```

68000      68020
add.w      d0,d0      move.w      (0,a1,d0.w*2),d1
move.w      (0,a1,d0.w),d1

```

16-bit Offsets on An+Rn Modes. The 68000 only supported 8-bit displacements when using the sum of an address register and another register as a memory address. The 68020 supports 16-bit displacements. This costs one extra cycle when the instruction is not in cache, but is free if the instruction is in cache. 32-bit displacements can also be used, but they cost 4 additional clock cycles.

Data Registers Used as Addresses. Register (d0) is 3 cycles slower than (a0), and it only takes 2 cycles to move a data register to an address register, but this can help in situations where there is no free address register.

Memory Indirect Addressing. These instructions can help in some circumstances when there are not any free registers to load a pointer into. Otherwise, they lose.

■ New Instructions

Extended Precision Divide and Multiply. The 68EC020 can perform 32x32->32, 32x32->64 multiplication and 32/32 and 64/32 division. These are significantly faster than the multi-precision operations which are required on the 68000.

EXTB. The sign extend byte to longword instruction is faster than the equivalent EXT.W EXT.L sequence on the 68000.

CMPI and TST. Compare immediate and TST work in program-counter relative mode on the 68020.

Shift Instructions. On the '020, all shift instructions execute in the same amount of time, regardless of how many bits are shifted. Note that ASL and ASR are slower than LSL and LSR. The break-even point on ADD Dn,Dn versus LSL is at two shifts.

Bit Field Instructions. BFINS inserts a bitfield, and is faster than 2 MOVEs plus an AND and an OR. This instruction can be used nicely in fill routines or text plotting. BFEXTU/BFEXTS can extract and optionally sign-extend a bitfield on an arbitrary boundary. BFFFO can find the highest order bit set in a field. BFSET, BFCHG, and BFCLR can set, complement, or clear up to 32 bits at arbitrary boundaries.

■ General Do's and Don'ts

- Do clear unused bits when writing, and mask out unused or unneeded bits when reading.
- Don't use timing loops. The reasons should be obvious.
- Don't write self-modifying code unless you know how instruction caches work.
- If you are hardware banging, don't assume anything about the initial contents of the display registers (or any other registers) when your program is started. Initialize everything.
- If using ViewPorts, be sure to have a properly allocated ViewPortExtra. Some graphics calls are faster when one is present.
- Do note well the warning around the copinit structure.

Conclusion

Writing OS-friendly games benefits you, the developer, and the customers who buy your games. It ensures the viability of your code on all Amiga platforms, not just the one you used for development. By allowing the OS to do the mundane details of opening screens and windows, you give yourself more time to do the creative portions of your game. It does not matter how fast your game is if the gameplay is lousy and the artwork looks amateurish.

An OS-friendly game is no longer something to be avoided. We've provided ample tools in the OS to support your gaming needs, and we will continue to do so. To get you started, the examples on the following pages show you how to set up your own View and ViewPort in both C and assembly language. The sooner you write OS-friendly games, the sooner you'll be prepared for the future.

LowLevelView.c

```

/* LowLevelView.c - Execute me to compile me with Lattice 6.2
sc data-near nominc strmer streq nostckchk saved ign-73 LowLevelView
quit
*/

#include <exec/memory.h>
#include <exec/libraries.h>
#include <graphics/view.h>
#include <graphics/gfxbase.h>
#include <graphics/videocontrol.h>
#include <graphics/monitor.h>
#include <graphics/displayinfo.h>
#include <hardware/custom.h>
#include <graphics/gfxmacros.h>
#include <clib/exec_protos.h>
#include <clib/graphics_protos.h>
extern struct GfxBase *GfxBase;
extern struct Library *IntuitionBase;
extern far struct Custom custom;

struct View *CreateAView(APTR, ULONG);
void DeleteAView(struct View *);
struct ViewPort *CreateAViewPort(APTR, ULONG, ULONG, ULONG, ULONG, ULONG);
void DeleteAViewPort(struct ViewPort *);

struct View *CreateAView(APTR mem_pool, ULONG modeid)
{
    struct ViewExtra *ve;
    struct View *view = NULL;

    if (view = AllocPooled(mem_pool, sizeof(struct View)))
    {
        InitView(view);

        if ((ve = GfxNew(VIEW_EXTRA_TYPE)) /* For V36 and up, you need a ViewExtra if
        /* if you want to use a non-default monitor. */
        {
            if ((ve->Monitor = OpenMonitor(NULL, modeid))
            {
                GfxAssociate(view, ve);
                view->Modes |= EXTEND_VSTRUCT; /* Mark the View as having a ViewExtra. */
            }
            else
            {
                GfxFree(ve);
                ve = NULL;
            }
        }
        if (!ve)
        {
            view = NULL;
        }
    }
    return(view);
}

void DeleteAView(struct View *view)
{
    struct ViewExtra *ve=NULL;
    if (ve = GfxLookup(view))
    {
        if (ve->Monitor) CloseMonitor(ve->Monitor);
        GfxFree(ve);
    }
}

```

LowLevelView.c

```

struct ViewPort *CreateAViewPort(APTR mem_pool,
    ULONG size,
    ULONG size,
    ULONG depth,
    ULONG modeid,
    ULONG colors)
{
    struct ViewPort *vp
    struct ViewPortExtra *vpextra = NULL;
    struct DisplayInfo *disinfo = NULL;

    if ((1L<depth) <= colors) /* There must be enough colors in the
    /* colormap for the ViewPort's bitmap. */
    {
        if (vp = AllocPooled(mem_pool, sizeof(struct ViewPort)))
        {
            InitVPort(vp); /* This function clears some of the ViewPort field so
            /* call it before initializing any ViewPort fields. */
            if (vp->ColorMap = GetColorMap(colors)) /* I need a ColorMap if I want to
            /* use certain VideoControl() features. */
            {
                if (vp->RasInfo = (struct RasInfo *)
                AllocPooled(mem_pool, sizeof(struct RasInfo)))
                {
                    if (vp->RasInfo->BitMap =
                    AllocBitMap(size, size, depth,
                    BMF_DISPLAYABLE | BMF_CLEAR,
                    NULL))
                    {
                        if (vpextra = GfxNew(VIEWPORT_EXTRA_TYPE))
                        {
                            if (disinfo = FindDisplayInfo(modeid))
                            {
                                vp->DWidth = size;
                                vp->DHeight = size;
                                if (!VideoControlTags(vp->ColorMap,
                                /* This tag associates a ColorMap with a ViewPort.
                                VTAG_ATTACH_CM_SET, vp,
                                /* This tag associates a ViewPortExtra with a ViewPort.
                                /* Notice that VideoControl() (*not* GfxAssociate())
                                /* associates the VP and VPE.
                                VTAG_VIEWPORTEXTRA_SET, vpextra,
                                VTAG_NORMAL_DISP_SET, disinfo,
                                VTAG_USERCLIP_SET, NULL,
                                TAG_END))
                                {
                                    if (AttachPalExtra(vp->ColorMap, vp))
                                    {
                                        disinfo = NULL;
                                    }
                                }
                                if (!disinfo) { GfxFree(vpextra); vpextra = NULL; }
                                if (!vpextra)
                                {
                                    FreeBitMap(vp->RasInfo->BitMap);
                                    vp->RasInfo->BitMap=NULL;
                                }
                                if (!((vp->RasInfo->BitMap) { vp->RasInfo=NULL; }
                                {
                                    if (!((vp->RasInfo) { FreeColorMap(vp->ColorMap); vp->ColorMap=NULL; }
                                    {
                                        if (!((vp->ColorMap) { vp=NULL; }
                                    }
                                }
                                return(vp);
                            }
                        }
                    }
                }
            }
        }
    }
}

```

LowLevelView.c

```

void DeleteAViewPort(struct ViewPort *vp)
{
    struct TagItem ti[2];

    ti[0].ti_Tag = VTAG_VIEWPORTEXTRA_GET;
    ti[0].ti_Data = NULL;
    /* VideoControl() will write over this with */
    /* ViewPortExtra address. */

    ti[1].ti_Tag = VTAG_END_CM;
    VideoControl(vp->ColorMap, ti);
    /* Can't use VideoControlTags() here. */
    if (ti[0].ti_Data)
    {
        GfxFree((APTR)ti[0].ti_Data);
        /* Free the ViewPortExtra, if it exists. */
        FreeBitMap(vp->RasInfo->BitMap);
        FreeColorMap(vp->ColorMap);
    }
}

```

LowLevelView.h

```

#include <exec/types.h>
#include <graphics/view.h>
#include <graphics/videocontrol.h>
#include <graphics/monitor.h>
#include <graphics/modeid.h>
#include <graphics/displayinfo.h>

/*#define SIZEX 320
#define SIZEY 200
#define DEPTH 4
#define MODEID (NTSC_MONITOR_ID | LORES_KEY)*/
#define COLORS 32

struct View *CreateAView(APTR, ULONG);
void DeleteAView(struct View *);
struct ViewPort *CreateAViewPort(APTR, ULONG, ULONG, ULONG, ULONG);
void DeleteAViewPort(struct ViewPort *);

```


LLV.asm

```

MOVEA.L MOVEA.L
MOVE.L  _GfxBase,A6
        ;set up view address for InitView
        ;load graphics library
InitView
        MOVEM.L  _VIEW_EXTRA_TYPE,D0
        GfxNew
        ;Ask OS for a ViewExtra
        D0
        endCAV
        ;branch if GfxNew failed
OpenMon:
        MOVEA.L  D0,A3
        MOVE.L   #0,A1
        D2,D0
        OpenMonitor
        D0
        UnGfxNew
        ;clean up if if OpenMon failed
GfxAssociate:
        MOVE.L   D0,ve_Monitor(A3)
        MOVEA.L  A2,A0
        MOVEA.L  A3,A1
        GfxAssociate
        $EXTEND_VSTRUCT,v_Modes(A2) ;Set EXTEND_VSTRUCT in View->Modes
endCAV:
        MOVE.L   A2,D0
        (SP)+,D2/A2-A3/A6
        RTS
UnGfxNew:
        MOVEA.L  A3,A0
        GfxFree
        endCAV
_DeleteAView:
        MOVEM.L  A2/A6,-(SP)
        MOVE.L   _GfxBase,A6
        GfxLookUp
        D0
        endDAV
        MOVEA.L  D0,A2
        ve_Monitor(A2)
        endDAV
        ve_Monitor(A2),A0
        ; If there is an open MonitorSpec, close it
        CloseMonitor
        A2,A0
        GfxFree
        MOVEA.L  (SP)+,A2/A6
        RTS
_CreateAViewPort:
        MOVEM.L  D3/A2-A6,-(SP)
        MOVEA.L  A0,A3
        ;save mem_pool address for later
        #1,L,D0
        D2,D0
        D0,D7
        CMF.L   D0,D7
        BPL.B   #0,L,D0
        MOVE.L   #0,L,D0
        endCAVP
        4,W,A6
        MOVEA.L  $vp_SIZEOF,D0
        AllocPooled
        D0
        endCAVP
        attcmset_tag,A0
        D0,(A0)
        MOVEA.L  D0,A2
        LEA
        MOVEA.L  attcmset_tag,A0
        MOVEA.L  D0,A2

```

LLV.asm

```

MOVEA.L GfxBase,A6
MOVEA.L A2,A0
SYSCALL InitVPort
;
;D7 contains the number of colors
MOVE.L D7,D0
SYSCALL GetColorMap
TST.L D0
BEQ.W endCAVP

MOVE.L D0,vp_ColorMap(A2)
MOVEA.L D0,A5
;Store ColorMap pointer in VPort
;cache ColorMap

4.W,A6
MOVEA.L A3,A0
MOVE.L #ri_SIZEOF,D0
SYSCALL AllocPooled
MOVEA.L GfxBase,A6
TST.L D0
BEQ.W UnGetCM

;branch if Allocated failed
D0,ri_BitMap(A3)
;store RasiInfo pointer in VP
;cache RasiInfo pointer (finished
;w/mem_pool)
MOVEA.L D0,A3
D4,D0
D5,D1
#BNE DISPLAYABLE(BNE_CLEAR),D3
4.W,A0
MOVEA.L #ri_SIZEOF,D0
SYSCALL AllocPooled
MOVEA.L GfxBase,A6
TST.L D0
BEQ.W UnGetCM

D0,ri_BitMap(A3)
;Store BitMap pointer in cached rasiInfo
#VIEWPORT_EXTRA_TYPE,D0
GfxNew
D0
UnGetCM

LEA vpsset_tag,A0
D0,(A0)
;Store VPE in its VideoControl tag
;load modeld
D6,D0
FindDisplayInfo
D0
UnGetVPE

dispsset_tag,A0
D0,(A0)
;Store displayInfo in its VideoControl tag
;set up its DWidth/DHeight
D4,vp_DWidth(A2)
D5,vp_DHeight(A2)

A5,A0
;load cached CM
vidtags,A1
;load videocontrol() tags
D0
UnGetVPE

A5,A0
;copy CM
A2,A1
;copy VP
SYSCALL AttachPalExtra
D0
UnGetVPE
;Clean up if AttachPalExtra() returns an error

MOVE.L A2,D0
;Make the VP the return value
(SP)+,D3/A2-A6

endCAVP:
MOVEM.L rts
;Load VPortExtra address
;...and relinquish it
UnGetVPE:
LEA MOVEA.L A1,A0
SYSCALL GfxFree

```

LLV.asm

```

UnGetBM:
MOVEA.L ri_BitMap(A3),A0
SYSCALL FreeBitMap
;Load BitMap and relinquish

UnGetCM:
MOVE.L vp_ColorMap(A2),A0
SYSCALL FreeColorMap
;Load ColorMap
;...and relinquish
MOVE.L #0,L,D0
BRA.W endCAVP
;return value = NULL

_deleteAViewPort:
MOVEM.L A2/A6,-(SP)
MOVEA.L A0,A2
;cache VPort address

MOVE.L _CfxBase,A6
MOVEA.L vp_ColorMap(A2),A0
LEA getvpe_tag,A1
;Load ColorMap address

SYSCALL VideoControl
;find the VPort's VPortExtra with the
;VTAG_VIEWPORTEXTRA_GET tag

MOVE.L (getvpe_result),D0
TST.L D0
BEQ.W 2$
MOVEA.L D0,A0
SYSCALL GfxFree
;If there is a VPE, free it

2$:
MOVEA.L vp_RasiInfo(A2),A1
MOVEA.L ri_BitMap(A1),A0
;load the RasiInfo's BitMap...
SYSCALL FreeBitMap
;...and free it
MOVEA.L vp_ColorMap(A2),A0
SYSCALL FreeColorMap
;load and free ColorMap

endDAVP:
MOVEM.L (SP)+,A2/A6
RTS

section vidtags,data
;The tags for VideoControl from

CreateAViewPort
CNOP 0,4
vidtags:
dc.l VTAG_ATTACH_CM_SET
attcmset_tag: dc.l 0,VTAG_VIEWPORTEXTRA_SET
vpsset_tag: dc.l 0,VTAG_NORMAL_DISP_SET
dispsset_tag: dc.l 0,VTAG_USERCLIP_SET
uolipsset_tag: dc.l 0,VTAG_END

getvpe_tag: dc.l VTAG_VIEWPORTEXTRA_GET
getvpe_result: dc.l 0,VTAG_END_CM

```


LLV.h

```
#include <exec/types.h>
#include <graphics/view.h>
#include <graphics/videocontrol.h>
#include <graphics/monitor.h>
#include <graphics/modelid.h>
#include <graphics/displayinfo.h>

/*#define SIZEX 320
#define SIZEY 200
#define DEPTH 4
#define MODEID (NTSC_MONITOR_ID | SUPERHIRES_KEY)*/
#define COLORS 32

struct View * __asm CreateView(register __a0 APTR view,
                             register __d0 ULONG modelid);

void __asm DeleteView(register __a0 struct View *view);

struct ViewPort * __asm CreateViewPort(register __a0 APTR mem_pool,
                                       register __d4 ULONG sizeX,
                                       register __d5 ULONG sizeY,
                                       register __d2 ULONG depth,
                                       register __d6 ULONG modelid,
                                       register __d7 ULONG colors);

void __asm DeleteViewPort(register __a0 struct ViewPort *vp);
```


4

Includes and Autodocs

This section lists the Autodocs (function descriptions) and include (header) files for the new system software modules present in the Amiga CD³² game machine. The following V40 system modules are covered here:

- `lowlevel.library` Provides functions for timing, interrupts, reading the game controller, handling the keyboard and suspending the Amiga's multitasking OS.
- `nonvolatile.library` Interface to the nonvolatile RAM or equivalent storage. This allows players to quit a game and later resume where they left off, even if power was off in the interim.
- `cd.device` This is the lower-level interface to the CD-ROM subsystem. It follows the conventions of Amiga Exec devices.

The latest includes and Autodocs for all the other system modules are available directly from Commodore; contact your local support manager. For additional information, refer to the *Amiga Technical Reference Series* published by Addison-Wesley. The volumes in this series are as follows:

- *Amiga ROM Kernel Reference Manual: Includes and Autodocs*, 3rd edition, ISBN 0-201-56773-3 (\$38.95 US). These are the include files and Autodoc function descriptions that correspond to Release 2 (V37 of the operating system).
- *Amiga ROM Kernel Reference Manual: Libraries*, 3rd edition, ISBN 0-201-56774-1 (\$38.95 US). Tutorial information with lots of examples on how to use the Amiga's system of resident, re-entrant function libraries. Covers graphics, Intuition and Exec.
- *Amiga ROM Kernel Reference Manual: Devices*, 3rd edition, ISBN 0-201-56775-X (\$28.95 US). Tutorial information with lots of examples on how to control the Amiga's standard device interface and all the system devices. Also covers the IFF file sharing standard.
- *Amiga Hardware Reference Manual*, 3rd edition, ISBN 0-201-56776-8 (\$28.95 US). Detailed explanations of how the Amiga's hardware components and custom chips work.
- *Amiga User Interface Style Guide*, 3rd edition, ISBN 0-201-57757-7 (\$21.95 US). A functional specification for applications that use Amiga's powerful user interface facilities.

Detailed information on the Amiga's DOS is published by Bantam Books:

- *Amiga DOS Manual*, 3rd edition, ISBN 0-553-35403-5 (\$24.95 US). This contains the include file listings and Autodocs for the DOS subsystem in Release 2 (V37). Also has tutorial information and examples showing how to use DOS commands, scripts, etc.

lowlevel.library

AddKBInt	56
AddTimerInt	57
AddVBlankInt	57
ElapsedTime	57
GetKey	58
GetLanguageSelection	58
QueryKeys	59
ReadJoyPort	59
RemKBInt	60
RemTimerInt	60
RemVBlankInt	60
SetJoyPortAttrsA	60
StartTimerInt	61
StopTimerInt	61
SystemControlA	62

lowlevel.library

AddKBInt**NAME**

AddKBInt - adds a routine to the keyboard interrupt. (V40)

SYNOPSIS

```
intHandle = AddKBInt (intRoutine, intData);
D0          AC      A1
APTR AddKBInt (APTR, APTR);
```

FUNCTION

This routine extends the functionality of the keyboard interrupt to include intRoutine. Since this is an extension of the normal keyboard interrupt all of the keyboard handshaking is handled. The keyboard error codes are filtered out and not passed to intRoutine.

The routine is called whenever the user enters a key on the keyboard.

The routine is called from within an interrupt, so normal restrictions apply. The routine must preserve the following registers: A2, A3, A4, A7, D2-D7. Other registers are scratch, except for D0, which MUST BE SET TO 0 upon exit. On entry to the routine, A1 holds 'intData' and A5 holds 'intRoutine', and D0 contains the rawkey code read from the keyboard.

The routine is not called when a reset is received from the keyboard.

This is a low level function that does not fit the normal Amiga multitasking model. The interrupt installed will have no knowledge of which window/screen currently has input focus.

If your program is to exit without reboot, you MUST call RemKBInt() before exiting.

Only one interrupt routine may be added to the system. ALWAYS check the return value in case some other task has previously used this function.

INPUTS

intRoutine - the routine to invoke every vblank. This routine should be as short as possible to minimize its effect on overall system performance.

intData - data passed to the routine in register A1. If more than one long word of data is required this should be a pointer to a structure that contains the required data.

RESULT

intHandle - a handle used to manipulate the interrupt, or NULL if it was not possible to attach the routine.

SEE ALSO

RemKBInt()

AddTimerInt

NAME
AddTimerInt - adds an interrupt that is executed at regular intervals. (V40)

SYNOPSIS

```
intHandle = AddTimerInt(intRoutine, intData);
D0      A0      A1
APTR AddTimerInt (APTR, APTR);
```

FUNCTION

Calling this routine causes the system to allocate a CIA timer and set up 'intRoutine' to service any interrupts caused by the timer. Although the timer is allocated it is neither running, nor enabled. StartTimer() must be called to establish the time interval and start the timer.

The routine is called from within an interrupt, so normal restrictions apply. The routine must preserve the following registers: A2, A3, A4, A7, D2-D7. Other registers are scratch, except for D0, which MUST BE SET TO 0 upon exit. On entry to the routine, A1 holds 'intData' and A5 holds 'intRoutine'.

Only a single CIA timer will be allocated by this routine. So this routine may only be called once without an intervening call to RemTimerInt().

The CIA timer used by this routine is not guaranteed to always be the same. This routine utilizes the CIA resource and uses an unallocated CIA timer.

If your program is to exit without reboot, you MUST match all calls to this function with calls to RemTimerInt() before exiting.

Even if you only use the function once in your program; checking the return value will make your program more tolerant for multitasking on the Amiga computer platforms.

INPUTS

intRoutine - the routine to invoke upon timer interrupts. This routine should be as short as possible to minimize its effect on overall system performance.
 intData - data passed to the routine in register A1. If more than one long word of data is required this should be a pointer to a structure that contains the required data.

RESULT

intHandle - a handle used to manipulate the interrupt, or NULL if it was not possible to attach the routine.

SEE ALSO

RemTimerInt(), StopTimerInt(), StartTimerInt()

AddVBlankInt

NAME
AddVBlankInt - adds a routine executed every vertical blank. (V40)

SYNOPSIS

```
intHandle = AddVBlankInt(intRoutine, intData);
D0      A0
APTR AddVBlankInt (APTR, APTR);
```

FUNCTION

Lets you attach a routine to the system which will get called everytime a vertical blanking interrupt occurs.

The routine is called from within an interrupt, so normal restrictions apply. The routine must preserve the following registers: A2, A3, A4, A7, D2-D7. Other registers are scratch, except for D0, which MUST BE SET TO 0 upon exit. On entry to the routine, A1 holds 'intData' and A5 holds 'intRoutine'.

If your program is to exit without reboot, you MUST call RemVBlankInt() before exiting.

Only one interrupt routine may be added to the system. ALWAYS check the return value in case some other task has previously used this function.

INPUTS

intRoutine - the routine to invoke every vblank. This routine should be as short as possible to minimize its effect on overall system performance.
 intData - data passed to the routine in register A1. If more than one long word of data is required this should be a pointer to a structure that contains the required data.

RESULT

intHandle - a handle used to manipulate the interrupt, or NULL if it was not possible to attach the routine.

SEE ALSO

RemVBlankInt()

ElapsedTime

NAME
ElapsedTime - returns the time elapsed since it was last called. (V40)

SYNOPSIS

```
fractionalSeconds = ElapsedTime(context);
D0      A0
ULONG ElapsedTime (struct EClockVal *);
```


lowlevel.library

FUNCTION

This function utilizes the timer.device/ReadEClock() function to get an accurate elapsed time value. Since the context needs to be established the first call to this routine will return a nonsense value.

The return value for this function only allows for sixteen bits worth for the integer number of seconds and sixteen bits for the fractional number of seconds.

With sixteen bits worth of integer seconds this function can be used to timer an interval up to about 16 hours. If the actual time interval is larger this function will return this maximum value.

The sixteen bits for fractional seconds gives a resolution of approximately 20 microseconds. However, it is not recommended to expect this function to be accurate for a time interval of less than 200 microseconds.

INPUTS

context - pointer to an EClockVal structure. The first time you call this function, you should initialize the structure to 0s. You should then reuse the same structure for subsequent calls to this function, as this is how the elapsed time is calculated.

RESULT

fractionalSeconds - The elapsed time as a fixed point 32-bit number with the point fixed in the middle. That is, the upper order sixteen bits represent the number of seconds elapsed. The low order sixteen bits represent the fractional number of seconds elapsed. This value is limited to about sixteen hours. Although this value is precise to nearly 20 microseconds it is only accurate to within 200 microseconds.

WARNING

The first call to this function will return a nonsense value. Only rely on its result starting with the second call.

SEE ALSO

timer.device/ReadEClock()

GetKey**NAME**

GetKey -- returns the currently pressed rawkey code and qualifiers. (V40)

SYNOPSIS

```
key = GetKey();
DO
    ULONG GetKey(VOID);
```

FUNCTION

This function returns the currently pressed non-qualifier key and all pressed qualifiers.

lowlevel.library

This function is safe within an interrupt.

This is a low level function that does not fit the normal Amiga multitasking model. The values returned by this function are not modified by which window/screen currently has input focus.

RESULT

key - key code for the last non-qualifier key pressed in the low order word. If no key is pressed this word will be FF. The upper order word contains the qualifiers which can be found within the long word as follows:

Qualifier	Key
LLKB_LSHIFT	Left Shift
LLKB_RSHIFT	Right Shift
LLKB_CAPSLOCK	Caps Lock
LLKB_CONTROL	Control
LLKB_LALT	Left Alt
LLKB_RALT	Right Alt
LLKB_LAMIGA	Left Amiga
LLKB_RAMIGA	Right Amiga

SEE ALSO

<libraries/lowlevel.h>

GetLanguageSelection**NAME**

GetLanguageSelection -- returns the current language selection. (V40)

SYNOPSIS

```
language = GetLanguageSelection();
DO
    ULONG GetLanguageSelection (VOID);
```

FUNCTION

Determine what the user has specified as a language.

RESULT

language - user specified language, or zero if none has yet been specified. See <libraries/lowlevel.h> for a definition of the currently supported language.

SEE ALSO

<libraries/lowlevel.h>, locale.doc

NAME _____

SYNOPSIS

```
VOID QueryKeys(struct KeyQuery *, UBYTE);
```

FUNCTION

This function may be invoked from within an interrupt, but the size of `QueryArray` should be kept as small as possible.

INPUTS

queryArray - an array of KeyQuery structures. The `kq_KeyCode` fields of these structures should be filled with the rawKey codes you wish to query about. Upon return from this function, the `kq_Pressed` field of these structures will be set to TRUE if the associated key is down, and FALSE if not.

arraySize - number of key code entries in queryArray

SEE ALSO

```
<libraries/lowlevel.h>
```

NAME _____

ReadJoyPort -- return the state of the selected joy/mouse port. (V40)

SYNOPSIS

```
portState = ReadJoyPort(portNumber);
```

ULONG ReadJoyPort (ULONG);

FUNCTION

This function is used to determine what device is attached to the joy port and the current position/button state. The user may attach a mouse, game controller, or joystick to the port and this function will dynamically detect which device is attached and return the appropriately formatted portState.

This function always executes immediately.

This is a low level single threaded function that does not fit the normal Amiga multitasking model. Only one task can be executing this routine at any time. All others will return immediately with IP TYPE NOTAVAIL.

The nature of this routine is not meant to encourage non-multitasking friendly programming practices like polling loops. If your tasks waiting for a transition to be returned use a `WaitTOF()` between calls to minimize the total system impact.

When called the first time, for each port, this function attempts to acquire certain system resources. In order to acquire these resources this function **MUST** be called from a task, or a DOS process. If this function fails to get the necessary resources, it will return with `JP_TYPE_NOTAVAIL`. Once the resources are acquired (return value other than `JP_TYPE_NOTAVAIL`) this function may be used in interrupts.

INPUTS

portNumber - port to read, in the range 0 to 3.

RESULT

portState - bit map that identifies the device and the current state of that device.
The format of the bit map is dependant on the type of device attached.

The following constants from `<libraries/lowlevel.h>` are used to determine which device is attached and the state of that device. The type of device can be determined by applying the mask `IP_TYPE_MASK` to the return value and comparing the resultant value with the following:

IP_TYPE_NOTAVAIL
IP_TYPE_GAMECTLR
IP_TYPE_MOUSE
IP_TYPE_JOYSTK
IP_TYPE_UNKNOWN

If type = JP_TYPE_GAMECTLR the bit map of portState is:

PPF BUTTON BLUE
 PPF BUTTON RED
 PPF BUTTON YELLOW
 PPF BUTTON GREEN
 PPF BUTTON FORWARD
 PPF BUTTON REVERSE
 PPF BUTTON PLAY
 PPF JOY UP
 PPF JOY DOWN
 PPF JOY LEFT
 PPF JOY RIGHT

lowlevel.library

If type = JP_TYPE_JOYSTICK the bit map of portState is:

```
JP_BUTTON_BLUE      Right
JP_BUTTON_RED       Fire
JP_JOY_UP           Up
JP_JOY_DOWN         Down
JP_JOY_LEFT         Left
JP_JOY_RIGHT        Right
```

If type = JP_TYPE_MOUSE the bit map of portState is:

```
JP_BUTTON_BLUE      Right mouse
JP_BUTTON_RED       Left mouse
JP_BUTTON_PLAY      Middle mouse
JP_MVERT_MASK       Mask for vertical counter
JP_MHORZ_MASK       Mask for horizontal counter
```

SEE ALSO

SetJoyPortAttrs()

RemKBIInt

NAME

RemKBIInt -- remove a previously installed keyboard interrupt. (V40)

SYNOPSIS

```
RemKBIInt (intHandle);
           AI
VOID RemKBIInt (APTR);
```

FUNCTION

Remove a keyboard interrupt routine previously added with AddKBIInt().

INPUTS

intHandle - handle obtained from AddKBIInt(). This may be NULL, in which case this function does nothing.

SEE ALSO

AddKBIInt()

RemTimerInt

NAME

RemTimerInt -- remove a previously installed timer interrupt. (V40)

SYNOPSIS

```
RemTimerInt (intHandle);
           AI
VOID RemTimerInt (APTR);
```

lowlevel.library

FUNCTION

Removes a timer interrupt routine previously installed with AddTimerInt.

INPUTS

intHandle - handle obtained from AddTimerInt(). This may be NULL, in which case this function does nothing.

SEE ALSO

AddTimerInt(), StopTimerInt(), StartTimerInt()

RemVBlankInt

NAME

RemVBlankInt -- remove a previously installed vertical blank routine. (V40)

SYNOPSIS

```
RemVBlankInt (intHandle);
           AI
VOID RemVBlankInt (APTR);
```

FUNCTION

Removes a vertical blank interrupt routine previously added with AddVBlankInt().

INPUTS

intHandle - handle obtained from AddVBlankInt(). This may be NULL, in which case this function does nothing.

SEE ALSO

AddVBlankInt()

SetJoyPortAttrs

NAME

SetJoyPortAttrsA -- change the attributes of a port. (V40.27)
SetJoyPortAttrs -- varargs stub for SetJoyPortAttrsA. (V40.27)

SYNOPSIS

```
success = SetJoyPortAttrsA(portNumber, tagList);
           DO AI
BOOL SetJoyPortAttrsA(ULONG, struct TagItem *);
Success = SetJoyPortAttrs(portNumber, firstTag, ...);
BOOL SetJoyPortAttrs(Tag, ...);
```


FUNCTION

This function allows modification of several attributes held by ReadJoyPort() about both it's operation and the type of controller currently plugged into the port.

ReadJoyPort()'s default behavior is to attempt to automatically sense the type of controller plugged into any given port, when asked to read that port. This behavior is beneficial, to allow simple detection of the type of controller plugged into the port. Unfortunately, rare cases are possible where extremely fine mouse movements appear to be real joystick movements. Also, this ability to auto-sense the controller type causes most reads to take longer than if there were no auto-sensing.

SetJoyPortAttrs() is intended to provide for both of these cases. It allows the programmer to notify ReadJoyPort() to stop spending time attempting to sense which type of controller is in use - and, optionally, to force ReadJoyPort() into utilizing a certain controller type.

INPUTS

portNumber - the joyport in question (0-3).
tagList - a pointer to an array of tags providing parameters to SetJoyPortAttrs(); if NULL, the function will return TRUE, but do nothing.

TAGS

SJA_Type (ULONG) - Sets the current controller type to the mouse, joystick, or game controller. Supply one of SJA_Type_GAMECTLR, SJA_Type_MOUSE, SJA_Type_JOYSTK, or SJA_Type_AUTOSENSE. If SJA_Type_AUTOSENSE is used, ReadJoyPort() will attempt to determine the type of controller plugged into the given port automatically. If one of the other types is used, ReadJoyPort() will make no attempt to read the given controller as anything other than the type specified. The default type is SJA_Type_AUTOSENSE.

Note - if you set the type to anything other than auto-sense, it's your responsibility to return it to auto-sense mode before exiting.

SJA_Reinitialize (VOID) - Return a given port to it's initial state, forcing a port to deallocate any allocated resources; return the implied type to SJA_Type_AUTOSENSE.

RESULT

success - TRUE if everything went according to plan, or FALSE upon failure

SEE ALSO

ReadJoyPort(), <libraries/lowlevel.h>

StartTimerInt**NAME**

StartTimerInt -- start the timer associated with the timer interrupt. (V40)

SYNOPSIS

```
StartTimerInt (intHandle, timeInterval, continuous);
               AI      DO
VOID StartTimerInt (APTR, ULONG, BOOL);
```

FUNCTION

This routine starts a stopped timer that is associated with a timer interrupt created by AddTimerInt().

INPUTS

intHandle - handle obtained from AddTimerInt().
timeInterval - number of microseconds between interrupts. The maximum value allowed is 90,000. If higher values are passed there will be unexpected results.
continuous - FALSE for a one shot interrupt. TRUE for multiple interrupts.

SEE ALSO

AddTimerInt(), RemTimerInt(), StopTimerInt()

StopTimerInt**NAME**

StopTimerInt -- stop the timer associated with the timer interrupt. (V40)

SYNOPSIS

```
StopTimerInt (intHandle);
              AI
VOID StopTimerInt (APTR);
```

FUNCTION

Stops the timer associated with the timer interrupt handle passed. This is used to stop a continuous timer started by StartTimerInt().

INPUTS

intHandle - handle obtained from AddTimerInt().

SEE ALSO

AddTimerInt(), RemTimerInt(), StartTimerInt()

lowlevel.library

SystemControlA

NAME

SystemControlA - Method for selectively disabling OS features. (V40)
SystemControl - varargs stub for SystemControlA().

SYNOPSIS

```
failTag = SystemControlA(tagList);
DO
    AI
ULONG SystemControlA(struct TagItem *);
failTag = SystemControl(firstTag, ...);
ULONG SystemControl(Tag, ...);
```

FUNCTION

This function is used to alter the operation of the system. Some of the alterations involve controlling what are normally regarded as system resources. In order to minimize confusion only one task is allowed to control any part of the system resources. This prevents the possibility of two tasks fighting, each controlling a part of the system. If a tag is identified as task exclusive, it means that only one task can hold (set to TRUE) that tag. If another task attempts to set the same tag to TRUE, the call to SystemControl() will fail.

It is important to remember that SystemControl() can fail.

This is a low level function and certain tags do not fit the normal Amiga multitasking model.

INPUTS

tagList - pointer to an array of tags listing the features of the system to be enabled/disabled.

TAGS

SCON_TakeOverSys (BOOL)

TRUE - Takes over the CPU to ensure that a program gets every ounce of CPU time (with the exception of crucial interrupts). When in this mode, the CPU will belong completely to the program. Task switching will be disabled and the program will get all CPU cycles. This means any calls to the OS that involve multitasking in some way will not execute correctly. Other tasks will not run until this tag is used with FALSE. However, during a Wait() on a signal, multitasking will automatically be turned back on until the signal is received. Once received, multitasking will again be disabled and the CPU will be exclusive to the owning program.

FALSE - Relinquishes the CPU and reenables multitasking. This tag is task exclusive. This tag nests. A task may take over the CPU several times before relinquishing it.

lowlevel.library

SCON_KillReq (BOOL)

TRUE - Disables system requesters. Normally, all games should disable requesters. These are the reasons for NOT disabling system requesters:

- 1- No calls in the program will cause a system requester.
- 2- The only thing that could cause a requester to appear is the lack of a CD in the drive and SCON_CDReboot is set to CDReboot_On, therefore a requester can't appear.
- 3- All disk I/O is through nonvolatile.library (which disables requesters) or through CD with SCON_CDReboot set to CDReboot_On.

FALSE - Enables requesters for the program.

This tag nests. Tasks may disable requesters several times before enabling them. However, there must be a matching number of calls.

SCON_CDReboot (ULONG)

CDReboot_On - Ejecting the CD will cause a reboot of the system. Use this only if the program cannot deal with error conditions.
CDReboot_Off - Ejecting the CD will not cause a reboot of the system. Use this if the program needs to insert CDs while running.
CDReboot_Default - Restore the default reboot behavior for this system. This should be used upon exit, if this tag had been used to change the reboot behaviour. For the CD32 this value is synonymous with CDReboot_On. For Amiga computers this value is synonymous with CDReboot_Off.

Note that the default reboot behavior differs depending on the platform. If a program requires a specific behavior it must use this function to set the behavior. For example, a CD audio mixer would use this tag with the data CDReboot_Off. This will allow the changing of audio CDs on the game machine as well as Amiga computers.

If, however, there is no error detection code at all this tag should be used with the data CDReboot_On.

We hope that no program will require CDReboot_On. If all programs check for error condition and recover gracefully such a call should never be necessary. With the default behavior the CD32 will always reset on disk ejects, and programs run from Amiga computers will not reset. Thus, leaving the default will increase the market for a program to include both types of platforms. This tag does not nest.

SCON_StopInput (BOOL) - When TRUE, stops input.device from using any CPU cycles. Also prevents input.device from passing along any events from either the keyboard and/or port 0.

This tag is task exclusive. This tag is NOT reversible. Attempting to reverse will result in confused/garbled input events.

lowlevel.library

SCON_AddCreateKeys (ULONG) - Starts creating rawkey codes for the joystick/game controller on the given unit. The unit value is checked for validity and must be either 0 or 1. Each different unit used results in some code added to the VBlank interrupt chain. This tag nests. The tag **SCON_RemCreateKeys** is used to undo this tag. Tasks may create rawkey codes several times before stopping them.

Note that when operating in an Intuition window, the controller's blue button is the equivalent of the mouse menu button. Therefore, Intuition will be capturing most blue button events. If notification of these events is important, review the documentation for **WFLG_RMBTRAP** in the `intuition.library/OpenWindow0` autodoc.

SCON_RemCreateKeys (ULONG) - stops rawkey codes for the joystick/game controller on the given unit. The unit value is checked for validity and must be either 0 or 1.

RESULT

failTag - zero if all tags succeeded. A non-zero return indicates a tag that has failed. It is possible that other tags may fail as well.

If any tag fails there will be no change in the system due to other tags.

SEE ALSO

<libraries/lowlevel.h>

nonvolatile.library

background

DeleteNV

FreeNVData

GetCopyNV

GetNVInfo

GetNVList

SetNVProtection

StoreNV

64

65

65

65

66

66

67

67

nonvolatile.library

background

PURPOSE

The nonvolatile library provides a simple means for an application developer to manage nonvolatile storage.

OVERVIEW

The nonvolatile library is meant to be used transparently across all configurations. Currently, nonvolatile storage may consist of NVRAM and/or disk devices. nonvolatile.library will automatically access the best nonvolatile storage available in the system. Disk based storage will be selected first and if not available, NVRAM storage will be accessed.

NVRAM. On low-end diskless Amiga platforms, NVRAM may be available. This RAM will maintain its data contents when the system is powered down. This is regardless of whether batteries or battery-backed clock are present. The data stored in NVRAM is accessible only through the ROM-based nonvolatile library function calls. The size of NVRAM storage is dependant on the system platform and is attainable through the GetNVInfo() function.

Disk. In keeping with the general configurability of the Amiga, the actual disk location used by nonvolatile library when storing to disk may be changed by the user.

The prefs directory is used on the Amiga for storing many user configurable options. The location for nonvolatile disk storage is contained in the file prefs/env-archive/sys/nv_location. This file should contain a data string that specifies a lockable location. If the string does not specify a lockable location, the file will be ignored.

When opened, the nonvolatile library will search all drives within the system until it finds this file and successfully accomplishes a Lock on the location specified in the file. To force a rescan of all drives, the library may be closed and reopened or execute the GetNVInfo() function.

DeleteNV**NAME**

DeleteNV -- remove an entry from nonvolatile storage. (V40)

SYNOPSIS

```
success = DeleteNV(appName, itemName, killRequesters);
D0      A0      A1      D1
```

```
BOOL DeleteNV (STRPTR, STRPTR, BOOL);
```

FUNCTION

Searches the nonvolatile storage for the indicated entry and removes it.

The strings appName and itemName may not contain the '/' or ':' characters. It is recommended that these characters be blocked from user input when requesting AppName and ItemName strings.

INPUTS

appName - NULL terminated string identifying the application that created the data. Maximum length is 31.

itemName - NULL terminated string uniquely identifying the data within the application. Maximum length is 31.

killRequesters - suppress system requesters flag. TRUE if all system requesters are to be suppressed during this function. FALSE if system requesters are allowed.

RESULT

success - TRUE will be returned if the entry is found and deleted. If the entry is not found, FALSE will be returned.

FreeNVData**NAME**

FreeNVData -- release the memory allocated by a function of this library. (V40)

SYNOPSIS

```
FreeNVData (data);
A0
```

```
VOID FreeNVData (APTR);
```

FUNCTION

Frees a block of memory that was allocated by any of the following:

GetCopyNV(), GetNVInfo(), GetNVList().

INPUTS

data - pointer to the memory block to be freed. If passed NULL, this function does nothing.

SEE ALSO

GetCopyNV(), GetNVInfo(), GetNVList()

GetCopyNV**NAME**

GetCopyNV -- return a copy of an item stored in nonvolatile storage. (V40)

SYNOPSIS

```
data = GetCopyNV (appName, itemName, killRequesters);
D0      A0      A1      D1
```

```
APTR GetCopyNV (STRPTR, STRPTR, BOOL);
```

FUNCTION

Searches the nonvolatile storage for the indicated appName and itemName. A pointer to a copy of this data will be returned.

The strings appName and itemName may not contain the '/' or ':' characters. It is recommended that these characters be blocked from user input when requesting appName and itemName strings.

INPUTS

appName - NULL terminated string indicating the application name to be found. Maximum length is 31.

itemName - NULL terminated string indicated the item within the application to be found. Maximum length is 31.

killRequesters - Suppress system requesters flag. TRUE if all system requesters are to be suppressed during this function. FALSE if system requesters are allowed.

RESULT

data - pointer to a copy of the data found in the nonvolatile storage associated with appName and itemName. NULL will be returned if there is insufficient memory or the appName/itemName does not exist.

SEE ALSO

FreeNVData(), <libraries/nonvolatile.h>

nonvolatile.library

GetNVInfo**NAME**

GetNVInfo -- report information on the current nonvolatile storage. (V40)

SYNOPSIS

```
Information = GetNVInfo(killRequesters);
D0
D1
struct NVInfo *GetNVInfo (BOOL);
```

FUNCTION

Finds the user's preferred nonvolatile device and reports information about it.

INPUTS

killRequesters - suppress system requesters flag. TRUE if all system requesters are to be suppressed during this function. FALSE if system requesters are allowed.

RESULT

information - pointer to an NVInfo structure. This structure contains information on the NV storage media with the largest storage. The structure contains 2 longword fields: nvi_MaxStorage and nvi_FreeStorage. Both values are rounded down to the nearest ten.

The nvi_MaxStorage field is defined as the total amount of nonvolatile storage available on this device. The nvi_FreeStorage is defined as the amount of available space for NVDISK or the amount of non-locked storage for NVRAM. For NVDISK, the nvi_FreeStorage takes into account the amount of overhead room required to store a new App/Item. This amount is 3 blocks to allow room for storing a new Item file and possibly a new App directory. For NVRAM, the amount of overhead is 5 bytes. However, the amount of room required to store a new NVRAM item depends on the length of the App and Item names. Refer to StoreNV0 function for storage details.

This function may return NULL in the case of failure.

SEE ALSO

FreeNVData0, StoreNV0, <libraries/nonvolatile.h>

nonvolatile.library

GetNVList**NAME**

GetNVList -- return a list of the items stored in nonvolatile storage. (V40)

SYNOPSIS

```
list = GetNVList(appName, killRequesters);
D0
D1
struct MinList *GetNVList (STRPTR, BOOL);
```

FUNCTION

Returns a pointer to an Exec list of nonvolatile Items associated with the appName requested.

The string appName may not contain the '/' or ':' characters. It is recommended that these characters be blocked from user input when requesting an appName string.

INPUTS

appName - NULL terminated string indicating the application name to be matched. Maximum length is 31.

killRequesters - Suppress system requesters flag. TRUE if all system requesters are to be suppressed during this function. FALSE if system requesters are allowed.

RESULT

list - a pointer to an Exec MinList of NVEntries. A NULL will be returned if there is insufficient memory. If there are no entries in the nonvolatile storage for the appName, an empty list will be returned.

NOTE

The protection field contains more bits than are required for storing the delete protection status. These bits are reserved for other system usage and may not be zero. When checking for the delete status use either the field mask NVIF_DELETE, or the bit definition NVIB_DELETE.

SEE ALSO

FreeNVData0, SetNVProtection0

SetNVProtection**NAME**

SetNVProtection - set the protection flags. (V40)

SYNOPSIS

```
success = SetNVProtection(appName, itemName, mask, killRequesters);
D0      A0      A1      D2      D1
```

```
BOOL SetNVProtection(STRPTR, STRPTR, LONG, BOOL);
```

FUNCTION

Sets the protection attributes for an item currently in the nonvolatile storage.

Although 'mask' is LONG only the delete bit, NVEF_DELETE/NVEB_DELETE, may be set. If any other bits are set this function will return FALSE.

The strings appName and itemName may not contain the '/' or ':' characters. It is recommended that these characters be blocked from user input when requesting AppName and ItemName strings.

INPUTS

appName - NULL terminated string indicating the application name to be matched. Maximum length is 31.

itemName - NULL terminated string indicated the item within the application to be found. Maximum length is 31.

mask - the new protection mask. Only set the delete bit otherwise this function WILL CRASH.

killRequesters - suppress system requesters flag. TRUE if all system requesters are to be suppressed during this function. FALSE if system requesters are allowed.

RESULT

success - FALSE if the protection could not be change (ie the data does not exist).

SEE ALSO

GetNVList0, <libraries/nonvolatile.h>

StoreNV**NAME**

StoreNV - store data in nonvolatile storage. (V40)

SYNOPSIS

```
error = StoreNV(appName, itemName, data, length, killRequesters);
D0      A0      A1      A2      D0      D1
```

```
UWORD StoreNV(STRPTR, STRPTR, APTR, ULONG, BOOL);
```

FUNCTION

Saves some data in nonvolatile storage. The data is tagged with AppName and ItemName so it can be retrieved later. No single item should be larger than one fourth of the maximum storage as returned by GetNVInfo0. There is no data compression associated with this function.

The strings AppName and ItemName, should be short, but descriptive. They need to be short since the string is stored with the data and the nonvolatile storage for a stand alone game system is limited. The game system allows the user to selectively remove entries from storage, so the string should be descriptive.

The strings AppName and ItemName may not contain the '/' or ':' characters. It is recommended that these characters be blocked from user input when requesting AppName and ItemName strings.

INPUTS

appName - NULL terminated string identifying the application creating the data. Maximum length is 31.

itemName - NULL terminated string uniquely identifying the data within the application. Maximum length is 31.

data - pointer to the memory block to be stored.

length - number of bytes to be stored in the units of tens of bytes. For example, if you have 23 bytes to store length = 3; 147 byte then length = 15.

killRequesters - suppress system requesters flag. TRUE if all system requesters are to be suppressed during this function. FALSE if system requesters are allowed.

RESULT

error - 0

means no error,

NVERR_BADNAME error in AppName, or ItemName.

NVERR_WRITEPROT Nonvolatile storage is read only.

NVERR_FAIL Failure in writing data (nonvolatile storage full, or write protected).

NVERR_FATAL Fatal error when accessing nonvolatile storage, possible loss of previously saved nonvolatile data.

SEE ALSO

GetCopyNV0, GetNVInfo0

cd.device

CD_ADDCHANGEINT	68
CD_ADDFRAMEINT	69
CD_ATTENUATE	69
CD_CHANGENUM	70
CD_CHANGESTATE	70
CD_CONFIG	70
CD_EJECT	71
CD_GETGEOMETRY	71
CD_INFO	72
CD_MOTOR	72
CD_PAUSE	72
CD_PLAYLSN	73
CD_PLAYMSF	73
CD_PLAYTRACK	74
CD_PROTSTATUS	74
CD_QCODELSN	74
CD_QCODEMSF	75
CD_READ	75
CD_READXL	76
CD_REMCHANGEINT	77
CD_REMFRAMEINT	77
CD_SEARCH	78
CD_SEEK	78
CD_TOCLSN	78
CD_TOCMSF	79
CloseDevice	79
OpenDevice	79

cd.device

CD_ADDCHANGEINT

NAME

CD_ADDCHANGEINT -- add a disk change software interrupt handler.

FUNCTION

This command lets you add a software interrupt handler to the disk device that gets invoked whenever a disk insertion or removal occurs.

You must pass in a properly initialized Exec Interrupt structure and be prepared to deal with disk insertions/removals immediately. The interrupt is generated by the exec Cause function, so you must preserve A6.

To set up the handler, an Interrupt structure must be initialized. This structure is supplied as the io_Data to the CD_ADDCHANGEINT command. The handler then gets linked into the handler chain and gets invoked whenever a disk change happens. You must eventually remove the handler before you exit.

This command only returns when the handler is removed. That is, the device holds onto the IO request until the CD_REMCHANGEINT command is executed with that same IO request. Hence, you must use SendIO() with this command.

IO REQUEST INPUT

io_Device preset by the call to OpenDevice()
 io_Unit preset by the call to OpenDevice()
 io_Command CD_ADDCHANGEINT
 io_Length sizeof(struct Interrupt)
 io_Data pointer to Interrupt structure

IO REQUEST RESULT

io_Error 0 for success, or an error code as defined in <devices/cd.h>

SEE ALSO

CD_REMCHANGEINT, <devices/cd.h>, <exec/interrupts.h>,
 exec.library/Cause()

CD_ADDFRAMEINT

NAME CD_ADDFRAMEINT – add a CD-frame software interrupt handler.

IO REQUEST

io_Device preset by the call to OpenDevice()
 io_Unit preset by the call to OpenDevice()
 io_Command CD_ADDFRAMEINT
 io_Length sizeof(struct Interrupt)
 io_Data pointer to Interrupt structure

RESULTS

io_Error 0 for success, or an error code as defined in <devices/cd.h>

FUNCTION

This command lets you add a software interrupt handler to the disk device that gets invoked whenever a new frame is encountered while CD audio is being played.

You must pass in a properly initialized Exec Interrupt structure and be prepared to deal with frame interrupts immediately. The interrupt is generated by the exec Cause function, so you must preserve A6.

To set up the handler, an Interrupt structure must be initialized. This structure is supplied in io_Data of the CD_ADDFRAMEINT command. The handler then gets linked into the handler chain and gets invoked whenever a frame event occurs. You must eventually remove the handler before you exit.

This command only returns when the handler is removed. That is, the device holds onto the IO request until the CD_REMFRAMEINT command is executed with that same IO request. Hence, you must use SendIO() with this command.

NOTES

The interrupt handler can be added before or after a play command is sent. Interrupts will only be generated while CD audio is playing. Interrupts will not be generated when audio is paused.

SEE ALSO

CD_REMFRAMEINT, <devices/cd.h>, <exec/interrupts.h>, exec.library/Cause()

CD_ATTENUATE

NAME CD_ATTENUATE – Attenuate CD audio volume (immediately or gradually)

IO REQUEST

io_Device preset by the call to OpenDevice()
 io_Unit preset by the call to OpenDevice()
 io_Command CD_ATTENUATE
 io_Data NULL
 io_Length duration of volume fade in frames
 io_Offset target volume level (0 - 0x7FFF) (-1 = status only)

RESULTS

io_Error Returns an error if drive does not support attenuation
 io_Actual current volume level (fade may be monitored)

FUNCTION

This command will ramp the CD audio volume up or down from its current value to the value contained in io_Offset. The range is 0 (silence) to 0x7FFF (full volume). If -1 is specified as the target, the attenuation will not be modified; the current attenuation value will be returned in io_Actual.

io_Length contains the duration of the fade. In seconds, this is io_Length divided by the current frame rate (usually 75).

Note that this command returns before the fade has completed. Thus, once started, a fade cannot be aborted. You can, however, send a new CD_ATTENUATE command, which will immediately override any fade currently in progress. An io_Length of zero means attenuate immediately.

If a gradual attenuation command is sent before the play command, the fade will begin as soon as the play command is sent.

NOTES

This command has no effect on Amiga audio volume, only CD audio.

If the drive does not support volume attenuation, but does support mute, a value of under \$0800 should be considered mute, and equal to or above should be full volume. If chunky attenuation is supported, the drive should do the best it can. If the drive does not support volume attenuation at all, an error should be returned. Even if only mute is supported, if gradual attenuation is requested, the device should still emulate the fade command and mute based on the \$0800 boundary.

cd.device

CD_CHANGENUM

NAME
CD_CHANGENUM -- return the current value of the disk-change counter.

FUNCTION
This command returns the current value of the disk-change counter. The disk change counter is incremented each time a disk is inserted or removed from the cd unit.

IO REQUEST INPUT
io_Device preset by the call to OpenDevice()
io_Unit preset by the call to OpenDevice()
io_Command CD_CHANGENUM

IO REQUEST RESULT
io_Error - 0 for success, or an error code as defined in <devices/cd.h>
io_Actual - if io_Error is 0, this contains the current value of the disk-change counter.

CD_CHANGESTATE

NAME
CD_CHANGESTATE -- check if a "valid" disk is currently in a drive.

FUNCTION
This command checks to see if there is a "valid" disk in a drive.

IO REQUEST INPUT
io_Device preset by the call to OpenDevice()
io_Unit preset by the call to OpenDevice()
io_Command CD_CHANGESTATE

IO REQUEST RESULT
io_Error - 0 for success, or an error code as defined in <devices/cd.h>
io_Actual - 0 means there is a disk while anything else indicates there is no disk.

NOTES
A "valid" disk is a disk with a readable table of contents.

cd.device

CD_CONFIG

NAME
CD_CONFIG -- Set drive preferences

IO REQUEST
io_Device preset by the call to OpenDevice()
io_Unit preset by the call to OpenDevice()
io_Command CD_CONFIG
io_Data pointer to first entry of TagList
io_Length 0

RESULTS
io_Error 0 for success, or an error code as defined in <devices/cd.h>

FUNCTION
This command sets one or more of the configuration items. The configuration items are:

TAGCD_PLAYSPD Default: 75
TAGCD_READSPD Default: 75 (do not count on this)
TAGCD_READXLSPEED Default: 75
TAGCD_SECTORSIZE Default: 2048
TAGCD_XLECC Default: 1 (on)
TAGCD_EJECTRESET Default: can be 0 (off) or 1 (on)

The speed settings are described in the number of frames (sectors) per second. All CD-ROM drives are capable of the 75 frames/second rate. Some drives are capable of 150 frames/second, and some even more. To determine the maximum frame rate of the drive, use the CD_INFO command. Valid values for caddyless Commodore CD-ROM drives are 75 and 150 (normal speed and double speed). All other values are invalid. You should always make sure the drive is capable of the configuration you are requesting by either using the CD_INFO command, and/or by checking for an error condition after submitting your request.

There are three different types of CD-ROM sectors. Mode 1 sectors (2048 bytes), mode 2 form 1 sectors (2048 bytes), and mode 2 form 2 sectors (2328 bytes). Normally, disks are encoded in Mode 1 format. Mode 2 form 1 is basically the same as mode 1; however, the mode 2 form 2 sector format contains no CD-ROM error correction information. In order to read information encoded in this sector format, the drive's sector size must be configured to 2328 byte sectors.

Error correction (ECC) of the READXL command can be turned off or on with this command. Error correction can be implemented in either hardware or software (depending on the CD-ROM drive). When ECC is implemented in software, CPU

usage can become bursty. Errors rarely occur on CDs unless they have numerous scratches, but when they do occur, they will cause a loss of CPU bandwidth. When ECC is implemented in hardware, no CPU bandwidth is lost – in this case, ECC will always be on no matter how you configure the drive because it is free. The READXL command is used primarily for displaying movie-like data. In this case, speed is essential and data integrity is not; however, if the CPU is not being utilized during an XL animation there is no need to disable ECC (since the bandwidth is there to be used). The only time ECC should be disabled is when you are doing intense calculations in the background of a READXL command, AND your program is time-critical. Do not forget to change this back when you are done!

To make the computer reset when a CD is ejected (for an application that does not exit), use the TAGCD_EJECTRESET tag. When possible, titles should be able to exit cleanly back to Workbench. Error conditions should be monitored when doing disk I/O.

EXAMPLE

```
/* Configure ReadXL for double-speed reading and turn off ECC when */
/* the ReadXL command is used. */
struct TagItem ConfigList[] = {
    { TAGCD_READLSPEED, 150 },
    { TAGCD_XLECC, 0 },
    { TAG_END, 0 },
};

ior->io.Command = CD_CONFIG;
ior->io.Data = (APTR)&ConfigList;
ior->io.Length = 0;
DoIO(ior);
if (ior->io.Error) printf("Could not be configured");
```

NOTES

Setting the configuration will not modify the behavior of a read or play command already in progress.

This can be a very dangerous command. If for instance you set TAGCD_SECTORSIZE to 2328, you will no longer be able to read any data encoded at 2048 byte sectors (e.g. the file system will not be able to read the disk anymore). After you read any data stored with this sector format, you should immediately configure back to the original default value (even if the read failed – the disk could be removed in the middle of your read). You should NEVER use this command if you are not the exclusive owner of your disk.

BUGS

TAG_IGNORE, TAG_MORE, and TAG_SKIP do not work. Do not use these.

SEE ALSO

CD_INFO, <utility/tagitem.h>

CD_EJECT

NAME

CD_EJECT – Open or close the CD's drive door

IO REQUEST

io_Command	CD_EJECT
io_Data	NULL
io_Length	requested state of drive door (0 == close, 1 == open)
io_Offset	0

RESULTS

io_Error	0 for success, or an error code as defined in <devices/cd.h>
io_Actual	previous state of drive door

FUNCTION

This command causes the CD-ROM drive's door to open or close. The desired state of the drive door is placed in io_Length. The previous state of the drive door is returned in io_Actual.

CD_GETGEOMETRY

NAME

CD_GETGEOMETRY – return the geometry of the drive.

FUNCTION

This command returns a full set of information about the layout of the drive. The information is returned in the DriveGeometry structure pointed to by io_Data.

IO REQUEST INPUT

io_Device	preset by the call to OpenDevice0
io_Unit	preset by the call to OpenDevice0
io_Command	CD_GETGEOMETRY
io_Data	pointer to a DriveGeometry structure
io_Length	sizeof(struct DriveGeometry)

IO REQUEST RESULT

io_Error	- 0 for success, or an error code as defined in <devices/cd.h>
io_Actual	length of data transferred.

SEE ALSO

CD_GETNUMTRACKS, <devices/trackdisk.h>

cd.device

CD_INFO

NAME
CD_INFO -- Return information/status of device

IO REQUEST

io_Device preset by the call to OpenDevice()
io_Unit preset by the call to OpenDevice()
io_Command CD_INFO
io_Data pointer to CDInfo structure
io_Length sizeof(struct CDInfo)

RESULTS

io_Error 0 for success, or an error code as defined in <devices/cd.h>
io_Actual length of data transferred

FUNCTION

This command returns current configurations and status of the device driver.

EXAMPLE

```
struct CDInfo Info;
ior->io_Command = CD_INFO;          /* Retrieve drive info. */
ior->io_Data = (APTR)Info;           /* Here's where we want it */
ior->io_Length = sizeof(struct CDInfo); /* Return whole structure */
DoIo(ior);
if (!ior->io_Error) {                /* Command succeeded */
    if (Info.Status & CDSTSF_PLAYING) printf("Audio is playing");
    else                             printf("Audio not playing");
}
```

SEE ALSO

<devices/cd.h>

CD_MOTOR

NAME
CD_MOTOR -- control the on/off state of a drive motor.

FUNCTION

This command gives control over the spindle motor. The motor may be turned on or off.

If the motor is just being turned on, the device will delay the proper amount of time to allow the drive to come up to speed. Turning the motor on or off manually

cd.device

is not necessary, the device does this automatically if it receives a request when the motor is off.

IO REQUEST INPUT

io_Device preset by the call to OpenDevice()
io_Unit preset by the call to OpenDevice()
io_Command CD_MOTOR
io_Length the requested state of the motor, 0 to turn the motor off, and 1 to turn the motor on.

IO REQUEST RESULT

io_Error - 0 for success, or an error code as defined in <devices/cd.h>
io_Actual - if io_Error is 0 this contains the previous state of the drive motor.

CD_PAUSE

NAME
CD_PAUSE -- Pause or unPause play command.

IO REQUEST

io_Device preset by the call to OpenDevice()
io_Unit preset by the call to OpenDevice()
io_Command CD_PAUSE
io_Data NULL
io_Length pausemode : 1 = pause play; 0 = do not pause play;
io_Offset 0

RESULTS

io_Actual - if io_Error is 0, this contains the previous pause state.

FUNCTION

This command will place the CD in, or take the CD out of pause mode. The desired pause state is placed in io_Length. This command only effects play commands. When the audio is playing and the pausemode is set, this command will immediately pause the audio output suspending the play command until the play is unpaused. When audio is not playing and the pausemode is set, this command will set the pause mode (having no immediate effect). When a play command is submitted, the laser will seek to the appropriate position and pause at that spot. The play command will be suspended until the play is unpaused (or the play is aborted).

CD_PLAYLSN

NAME
CD_PLAYLSN -- Play a selected portion of CD audio (LSN form).

IO REQUEST
io_Device preset by the call to OpenDevice()
io_Unit preset by the call to OpenDevice()
io_Command CD_PLAYLSN
io_Data NULL
io_Length length of play
io_Offset starting position

RESULTS
io_Error 0 for success, or an error code as defined in <devices/cd.h>

FUNCTION

This command causes the drive to start playing CD audio from the specified position until the specified length has passed.
io_Offset specifies the starting position. io_Length contains the amount of time to play. All data is specified in LSN format.

A DoIO() will not return until the requested number of sectors have been played. A SendIO() will return as soon as the PLAY has been started. At this time other commands can be sent (like CD_PAUSE). To stop a play before the specified length has been reached, use AbortIO().

EXAMPLE

```
/* Play two minutes, ten seconds of audio starting at 20 minutes, */
/* 58 seconds, and 10 frames. */
ior->io_Command = CD_PLAYLSN; /* Play CD audio */
ior->io_Offset = 94360; /* 20*(60*75) + 58*75 + 10 */
ior->io_Length = 9750; /* 02*(60*75) + 10*75 + 00 */
DoIO (ior);
```

SEE ALSO

CD_PLAYTRACK, CD_PAUSE, CD_SEARCH, CD_ATTENUATE

CD_PLAYMSF

NAME
CD_PLAYMSF -- Play a selected portion of CD audio (MSF form).

IO REQUEST
io_Device preset by the call to OpenDevice()
io_Unit preset by the call to OpenDevice()
io_Command CD_PLAYMSF
io_Data NULL
io_Length length of play
io_Offset starting position

RESULTS
io_Error 0 for success, or an error code as defined in <devices/cd.h>

FUNCTION

This command causes the drive to start playing CD audio from the specified position until the specified length has passed.

io_Offset specifies the starting position. io_Length contains the amount of time to play. All data is specified in MSF format.

A DoIO() will not return until the requested number of sectors have been played. A SendIO() will return as soon as the PLAY has been started. At this time other commands can be sent (like CD_PAUSE). To stop a play before the specified length has been reached, use AbortIO().

EXAMPLE

```
/* Play two minutes, ten seconds of audio starting at 20 minutes, */
/* 58 seconds, and 10 frames. */
ior->io_Command = CD_PLAYMSF; /* Play CD audio */
ior->io_Offset = 0x0143A0A; /* $14=20, $3A=58, $0A=10 */
ior->io_Length = 0x00020A00; /* $02=02, $0A=10, $00=00 */
DoIO (ior);
```

SEE ALSO

CD_PLAYTRACK, CD_PAUSE, CD_SEARCH, CD_ATTENUATE

cd.device

CD_PLAYTRACK

NAME
CD_PLAYTRACK – Play one or more tracks of CD audio.

IO REQUEST

io_Device preset by the call to OpenDevice0
io_Unit preset by the call to OpenDevice0
io_Command CD_PLAYTRACK
io_Data NULL
io_Length number of tracks to play
io_Offset start playing at beginning of this track

RESULTS

io_Error 0 for success, or an error code as defined in <devices/cd.h>

FUNCTION

This command causes the drive to play the specified audio track(s). The command will return when the audio has completed.

io_Offset specifies the track number (starting from 1).

io_Length specifies the number of tracks to play (0 is invalid).

EXAMPLE

```
for->io_Command = CD_PLAYTRACK;      /* Play audio tracks */
for->io_Offset = STARTTRACK;        /* Start with this track */
for->io_Length = 3;                  /* Play three tracks */
DoIO(io_T);
```

NOTES

PLAY commands are asynchronous with many other CD commands. Using a separate I/O request, other commands can be sent to the device that can change the behavior of the PLAY command.

SEE ALSO

CD_PLAYMSF, CD_PLAYLSN, CD_PAUSE, CD_SEARCH, CD_ATTENUATE

cd.device

CD_PROTSTATUS

NAME
CD_PROTSTATUS – return whether the current disk is write-protected.

FUNCTION

This command is used to determine whether the current disk is write-protected. Currently, this function always returns write-protected status. If write-once CDs are made available at some point, this may change.

IO REQUEST INPUT

io_Device preset by the call to OpenDevice0
io_Unit preset by the call to OpenDevice0
io_Command CD_PROTSTATUS

IO REQUEST RESULT

io_Error - 0 for success, or an error code as defined in <devices/cd.h>
io_Actual - 0 means the disk is NOT write-protected, while any other value indicates it is.

CD_QCODELSN

NAME
CD_QCODELSN – Report current disk position.

IO REQUEST

io_Device preset by the call to OpenDevice0
io_Unit preset by the call to OpenDevice0
io_Command CD_QCODELSN
io_Data pointer to QCode structure
io_Length 0 - MUST be zero (for future compatibility)

RESULTS

io_Error 0 for success, or an error code as defined in <devices/cd.h>

FUNCTION

This command reports current subcode Q channel time information. This command only returns data when CD Audio is playing (or paused). At any other time, an error is returned. The Q-Code packet consists of:

cd.device

```
struct QCode {
    UBYTE CtlAdr;
    UBYTE Track;
    UBYTE Index;
    UBYTE Zero;
    union LSNMSF TrackPosition;
    union LSNMSF DiskPosition;
};

/* Data type / QCode type
/* Track number
/* Track subindex number
/* The "zero" byte of Q-Code packet
/* Position from start of track
/* Position from start of disk
```

EXAMPLE

```
struct QCode qcode;
for->io Command = CD_QCODELSN; /* Retrieve TOC information */
for->io Length = 0; /* MUST be zero
for->io Data = (APTR)qcode; /* Here's where we want it */
DoIO (for);

if (!for->io_Error) { /* Command succeeded */
    printf("Current position is: %ld", qcode.DiskPosition.LSN);
}
```

NOTES

This function may not return immediately. It may take several frames to pass by before a valid Q-Code packet can be returned. Use SendIO() and CheckIO() if response time is critical, and the information is not.

SEE ALSO

CD_PLAYMSF, CD_PLAYLSN, CD_PLAYTRACK, <devices/cd.h>

CD_QCODEMSF

NAME
CD_QCODEMSF – Report current disk position.

IO REQUEST

io_Device preset by the call to OpenDevice()
io_Unit preset by the call to OpenDevice()
io_Command CD_QCODEMSF
io_Data pointer to QCode structure
io_Length 0 - MUST be zero (for future compatibility)

RESULTS

io_Error 0 for success, or an error code as defined in <devices/cd.h>

FUNCTION

This command reports current subcode Q channel time information. This command only returns data when CD Audio is playing (or paused). At any other time, an error is returned. The Q-Code packet consists of:

cd.device

```
struct QCode {
    UBYTE CtlAdr;
    UBYTE Track;
    UBYTE Index;
    UBYTE Zero;
    union LSNMSF TrackPosition;
    union LSNMSF DiskPosition;
};

/* Data type / QCode type
/* Track number
/* Track subindex number
/* The "zero" byte of Q-Code packet
/* Position from start of track
/* Position from start of disk
```

EXAMPLE

```
struct QCode qcode;
for->io Command = CD_QCODEMSF; /* Retrieve TOC information */
for->io Length = 0; /* MUST be zero
for->io Data = (APTR)qcode; /* Here's where we want it */
DoIO (for);

if (!for->io_Error) { /* Command succeeded */
    printf("Current position is: %02d:%02d:%02d",
    qcode.DiskPosition.NSF.Minute,
    qcode.DiskPosition.NSF.Second,
    qcode.DiskPosition.NSF.Frame);
}
```

NOTES

This function may not return immediately. It may take several frames to pass by before a valid Q-Code packet can be returned. Use SendIO() and CheckIO() if response time is critical, and the information is not.

SEE ALSO

CD_PLAYMSF, CD_PLAYLSN, CD_PLAYTRACK, <devices/cd.h>

CD_READ

NAME
CD_READ – read data from disk.

FUNCTION

Reads data from the CD into memory. Data may be accessed on WORD boundaries (you are not restricted to sector boundaries as with normal disk devices). Data lengths can also be described in WORD amounts.

IO REQUEST INPUT

io_Device preset by the call to OpenDevice()
io_Unit preset by the call to OpenDevice()
io_Command CD_READ
io_Data pointer to the buffer where the data should be put
io_Length number of bytes to read, must be a WORD multiple.
io_Offset byte offset from the start of the disk describing where to read data from, must be a WORD multiple.

cd.device

IO REQUEST RESULT

io_Error - 0 for success, or an error code as defined in <devices/cd.h>
 io_Actual - if io_Error is 0, number of bytes actually transferred

NOTES

If an error occurs when attempting a CD_READ, the software will retry up to 10 times before giving up on the request. If the drive is in double-speed mode and an error occurs, the software will retry once more in double-speed and, if this fails, will retry the next 9 times in single-speed mode.

SEE ALSO

CD_READXL

CD_READXL

NAME

CD_READXL -- Read from CD-ROM into memory via transfer list.

IO REQUEST

io_Device preset by the call to OpenDevice()
 io_Unit preset by the call to OpenDevice()
 io_Command CD_READXL
 io_Data pointer to transfer list (i.e. struct List *).
 io_Length maximum transfer length (WORD multiple) or 0.
 io_Offset byte offset from the start of the disk describing where to read data from, must be a WORD multiple.

RESULTS

io_Error 0 for success, or an error code as described in <devices/cd.h>
 io_Actual if io_Error is 0, number of bytes actually transferred

FUNCTION

This command starts reading data off the disk at the specified location and deposits it into memory according to the nodes in a transfer list. The pointer to the list of transfer nodes is placed in io_Data. If you have a non-circular transfer list, simply set io_Length to 0 (0 is special and means ignore io_Length) -- your transfer will end when your transfer list has been exhausted. If you have a circular transfer list, the list will never end. In this case, the transfer will terminate when io_Length bytes have been transferred.

The fields in the CDXL node structure are:

```
struct CDXL {
  struct MinNode *Node; /* double linkage
  char *Buffer; /* data destination
  LONG Actual; /* must be reserved
  LONG ActData; /* bytes transferred
  APTR IntData; /* interrupt server data segment
  VOID (*IntCode)(); /* interrupt server code entry
};
```

cd.device

The philosophy here is that you set up the buffers you want filled, create CDXL nodes describing the locations and sizes of these buffers, link all the nodes together in the order that you'd like (even make a circular list for animations), and execute the command. The data will be streamed into the appropriate buffers until the list has been exhausted, an entry with a Length of zero is encountered, io_Length bytes have been transferred (if io_Length is non-zero), or the command is aborted with AbortIO().

If you fill in the (*IntCode) field with a pointer to an interrupt routine, your routine will be called when the transfer for the node is complete. Your code will be called before the driver proceeds to the next node. The interrupt should follow the same rules as standard interrupts (see AddIntServer of Exec autodecs). Register A2 will contain a pointer to the node just completed. You may manipulate the list from within the interrupt. Your code must be brief (this is an interrupt). When returning from this interrupt, D0 should be cleared and an RTS instruction should be used to return.

Servers are called with the following register conventions:

D0 - scratch
 D1 - scratch
 A0 - scratch
 A1 - server is Data pointer (scratch)
 A2 - pointer to CDXL node just completed
 A5 - jump vector register (scratch)

all other registers must be preserved

NOTES

Try to make sure that small buffers are not overused. Each time a node is completed, an interrupt is generated. If you find that your computer is acting sluggish, or the CD_READXL command is aborting, you are probably generating too many interrupts. It is not efficient to have more than a few of these interrupts generated within a vertical blank.

Unlike the READ command, the READXL command will not retry a sector if there is an error. Since the READXL command's purpose is primarily for animations, data streaming is considered more important than the data itself. An error will be returned in io_Error if a data error did occur. This command will never drop to a lower speed in the event of an error.

SEE ALSO

CMD_READ, CD_SEEK, Autodecs - AddIntServer

CD_REMCHANGEINT

NAME
CD_REMCHANGEINT -- remove a disk change software interrupt handler.

FUNCTION
This command removes a disk change software interrupt added by a previous use of CD_ADDCHANGEINT.

IO REQUEST INPUT
The same IO request used for CD_ADDCHANGEINT.

io_Device preset by the call to OpenDevice()
io_Unit preset by the call to OpenDevice()
io_Command CD_REMCHANGEINT
io_Length sizeof(struct Interrupt)
io_Data pointer to Interrupt structure

IO REQUEST RESULT
io_Error - 0 for success, or an error code as defined in <devices/cd.h>

SEE ALSO
CD_ADDCHANGEINT, <devices/cd.h>

CD_REMFRAMEINT

NAME
CD_REMFRAMEINT -- remove a CD-frame interrupt handler.

IO REQUEST
The same IO request used for CD_ADDFRAMEINT.

io_Device preset by the call to OpenDevice()
io_Unit preset by the call to OpenDevice()
io_Command CD_REMFRAMEINT
io_Length sizeof(struct Interrupt)
io_Data pointer to Interrupt structure

RESULTS
io_Error 0 for success, or an error code as defined in <devices/cd.h>

FUNCTION
This command removes a CD-frame software interrupt added by a previous use of CD_ADDFRAMEINT.

SEE ALSO
CD_ADDFRAMEINT, <devices/cd.h>

CD_SEARCH

NAME
CD_SEARCH -- configure the mode in which PLAY commands play

IO REQUEST
io_Command CD_SEARCH
io_Data NULL
io_Length searchmode
io_Offset 0

RESULTS
io_Actual - if io_Error is 0, this contains the previous search mode.

FUNCTION
This command causes a play command to play in fast-forward, fast-reverse, or normal play mode. These modes are defined as:

CDMODE_NORMAL 0 Normal play (current speed setting)
CDMODE_FFWD 1 Play in fast forward mode
CDMODE_FREV 2 Play in fast reverse mode

The search mode can be set before the play command is sent, or during a play. If CD_SEARCH is sent before a play command is sent, the mode is set and the command immediately returns. If the mode is set to REV mode, when the play command is sent the play will begin at the requested end position and work backwards to the start position.

If CD_SEARCH is sent during a play, the play will automatically switch to the desired mode and continue playing until the original play command is completed. If REV mode is set and the beginning of the play is encountered before switching back to forward play, the play command will terminate with no error.

EXAMPLE
/* Search in fast forward mode. */
for->io_Command = CD_SEARCH;
for->io_Data = NULL;
for->io_Offset = 0;
for->io_Length = CDMODE_FFWD;
DoIO(for);

cd.device

CD_SEEK

NAME

CD_SEEK -- position laser at specified location.

FUNCTION

CD_SEEK moves the laser to the approximate position specified. The io_Offset field should be set to the offset to which the head is to be positioned.

IO REQUEST INPUT

io_Device preset by the call to OpenDevice()
 io_Unit preset by the call to OpenDevice()
 io_Command CD_SEEK
 io_Offset position where head is to be moved (always LSN format)

IO REQUEST RESULT

io_Error - 0 for success, or an error code as defined in <devices/cd.h>

CD_TOCLSN

NAME

CD_TOCLSN -- Return table of contents information from CD (LSN form).

IO REQUEST

io_Device preset by the call to OpenDevice()
 io_Unit preset by the call to OpenDevice()
 io_Command CD_TOCLSN
 io_Data pointer to array where TOC is to be stored
 io_Length number of CDTOC entries to be fetched
 io_Offset entry to begin at (entry 0 is summary information)

RESULTS

io_Error 0 for success, or an error code as defined in <devices/cd.h>
 io_Actual Actual number of entries copied

FUNCTION

This command returns the table of contents of the disk currently in the drive. The table of contents consists of up to 100 entries. Entry zero is summary information describing the number of tracks and the total number of minutes on the disk. Entries 1 through N contain information about each individual track. All position information will be in LSN format.

cd.device

The io_Data field points to an array of CDTOC structures to receive the TOC data.

The io_Length field specifies the total number of entries to be fetched. The array pointed to by io_Data must be at least this many elements in size.

The io_Offset field specifies the entry number at which to start copying TOC data into *io_Data.

Entry zero (the summary entry) contains the following:

```
struct TOCSummary {
    UBYTE FirstTrack; /* First track on disk (always 1) */
    UBYTE LastTrack; /* Last track on disk */
    union LSNMSF LeadOut; /* Beginning of lead-out track */
};

Track entries (entries 1 through number of tracks) contain:

struct TOCEntry {
    UBYTE CtlAdr; /* Q-Code info */
    UBYTE Track; /* Track number */
    union LSNMSF Position; /* Start position of this track */
};
```

CDTOC is described as a union between these two structures:

```
union CDTOC {
    struct TOCSummary Summary; /* First entry is summary info. */
    struct TOCEntry Entry; /* Entries 1-N are track entries */
};
```

EXAMPLE

```
union CDTOC toccarray[100];

for->io_Command = CD_TOCLSN; /* Retrieve TOC information */
for->io_Offset = 0; /* Start with summary info */
for->io_Length = 100; /* Max 99 tracks + summary */
for->io_Data = (APTR) toccarray; /* Here's where we want it */
doIO (for);

if (!for->io_Error) { /* Command succeeded */
    firstTrack = toccarray[0].Summary.FirstTrack;
    lastTrack = toccarray[0].Summary.LastTrack;
    totalSectors = toccarray[0].Summary.Sectors;
    toccarray[1].Entry.Position.LSN -
}
```

NOTES

In the above example, the amount of data on the disk is calculated as being equal to the location of the lead-out track minus the start of the first track (which is never zero).

cd.device

CD_TOCMSF

NAME
CD_TOCMSF – Return table of contents information from CD (MSF form).

IO REQUEST

io_Device preset by the call to OpenDevice()
io_Unit preset by the call to OpenDevice()
io_Command CD_TOCMSF
io_Data pointer to array where TOC is to be stored
io_Length number of CDTOC entries to be fetched
io_Offset entry to begin at (entry 0 is summary information)

RESULTS

io_Error 0 for success, or an error code as defined in <devices/cd.h>
io_Actual Actual number of entries copied

FUNCTION

This command returns the table of contents of the disk currently in the drive. The table of contents consists of up to 100 entries. Entry zero is summary information describing the number of tracks and the total number of minutes on the disk. Entries 1 through N contain information about each individual track. All position information will be in MSF format.

The io_Data field points to an array of CDTOC structures to receive the TOC data.

The io_Length field specifies the total number of entries to be fetched. The array pointed to by io_Data must be at least this many elements in size.

The io_Offset field specifies the entry number at which to start copying TOC data into *io_Data.

Entry zero (the summary entry) contains the following:

```
struct TOCSummary {
    UBYTE FirstTrack; /* First track on disk (always 1) */
    UBYTE LastTrack; /* Last track on disk */
    union LSNMSF LeadOut; /* Beginning of lead-out track */
};
```

Track entries (entries 1 through number of tracks) contain:

```
struct TOCEntry {
    UBYTE CtlAddr; /* Q-Code info */
    UBYTE Track; /* Track number */
    union LSNMSF Position; /* Start position of this track */
};
```

cd.device

CDTOC is described as a union between these two structures:

```
union CDTOC {
    struct TOCSummary Summary; /* First entry is summary info. */
    struct TOCEntry Entry; /* Entries 1-N are track entries */
};
```

CloseDevice

NAME

CloseDevice - terminate access to the CD

SYNOPSIS

```
CloseDevice (IORequest);
            AI
```

FUNCTION

This function will terminate access to the unit opened with OpenDevice0.

INPUTS

ioRequest - pointer to a struct (IOStdReq)

SEE ALSO

OpenDevice0

OpenDevice

NAME

OpenDevice - Open a CD unit for access

SYNOPSIS

```
error = OpenDevice ("cd.device", UnitNumber, IORequest, flags);
            D0 AI D0 AI
```

FUNCTION

Opens the cd.device and creates an IORequest for use in accessing the CD.

INPUTS

UnitNumber - Normally zero; however, this is described as:
Ones digit = Unit (SCSI unit number)
Tens digit = LUN (disk within disk changer)
Hundreds digit= Card number (SCSI card)
Thousands digit= Reserved (must be zero)

cd.device

**IORequest
flags**

- Pointer to a struct(IOStdReq)
- Should be zero.

RESULTS

error

0 = success, otherwise this is an error.

SEE ALSO

CloseDevice()

INCLUDES

devices/cd.h
devices/cd.i

81
84

libraries/lowlevel.h
libraries/lowlevel.i

87
89

libraries/nonvolatile.h
libraries/nonvolatile.i

91
91

devices/cd.h

```
#ifndef DEVICES_CD_H
#define DEVICES_CD_H
/*
** $VER: cd.h 1.11 (12.8.93)
** Includes Release 40.13
**
** cd.device include file
**
** (C) Copyright 1992-1993 Commodore-Amiga, Inc.
** All Rights Reserved
**
*/
#include <exec/types.h>
#include <exec/nodes.h>

/*****
* * CD Commands
* * *****/

#define CD_RESET 1
#define CD_READ 2
#define CD_WRITE 3
#define CD_UPDATE 4
#define CD_CLEAR 5
#define CD_STOP 6
#define CD_START 7
#define CD_FLUSH 8
#define CD_MOTOR 9
#define CD_SEEK 10
#define CD_FORMAT 11
#define CD_REMOVE 12
#define CD_CHANGENUM 13
#define CD_CHANGEID 14
#define CD_PROTSTATUS 15

#define CD_GETDRIVETYPE 18
#define CD_GETNUMTRACKS 19
#define CD_ADCHANGEINT 20
#define CD_REMCHANGEINT 21
#define CD_GETGEOMETRY 22
#define CD_EJECT 23

#define CD_INFO 32
#define CD_CONFIG 33
#define CD_TOCMSF 34
#define CD_TOCLSN 35

#define CD_READXL 36

#define CD_PLAYTRACK 37
#define CD_PLAYMSF 38
#define CD_PLAYLSN 39
#define CD_PAUSE 40
#define CD_SEARCH 41

#define CD_QCODEMSF 42
#define CD_QCODELSN 43
#define CD_ATTENUATE 44

#define CD_ADDFRAMEINT 45
#define CD_REMFRAMEINT 46

/*****
* * Device Driver Error Codes
* * *****/

```

devices/cd.h

```

#define CDERR_OPENFAIL /* device/unit failed to open */
#define CDERR_ABORTED /* request terminated early */
#define CDERR_NOCD /* command not supported by device */
#define CDERR_BADLENGTH /* invalid length (IO LENGTH/IO OFFSET) */
#define CDERR_BADADDRESS /* invalid address (IO DATA misaligned) */
#define CDERR_UNRECOVERABLE /* device opens ok, but unit is busy */
#define CDERR_SECTEST /* hardware failed self-test */

#define CDERR_NotSpecified 20 /* general catchall */
#define CDERR_NoSector 21 /* couldn't even find a sector */
#define CDERR_BadSector 22 /* sector looked wrong */
#define CDERR_BadSecID 23 /* ditto */
#define CDERR_BadChecksum 24 /* header had incorrect checksum */
#define CDERR_BadDrumSum 25 /* data had incorrect checksum */
#define CDERR_TooFewSecs 26 /* couldn't find enough sectors */
#define CDERR_TooManySecs 27 /* couldn't find enough sectors */
#define CDERR_WriteProt 28 /* can't write to a protected disk */
#define CDERR_NoDisk 29 /* no disk in the drive */
#define CDERR_NoDisk2 30 /* couldn't find track 0 */
#define CDERR_NoMem 31 /* ran out of memory */
#define CDERR_NoUnit 32 /* asked for a unit > NUMUNITS */
#define CDERR_BadUnitNum 33 /* not a drive cd.device understands */
#define CDERR_DriveInUse 34 /* someone else allocated the drive */
#define CDERR_PostReset 35 /* user hit reset; awaiting doom */
#define CDERR_BadData 36 /* data on disk is wrong type */
#define CDERR_InvalidState 37 /* invalid cmd under current conditions */

#define CDERR_Phase 42 /* illegal or unexpected SCSI phase */
#define CDERR_NoBoard 50 /* open failed for non-existent board */

/* Configuration
 *
 * The drive is configured by TagList items defined as follows:
 */

#define TAGCD_PLAYSPD 0x0001
#define TAGCD_READSPD 0x0002
#define TAGCD_READLSPEED 0x0003
#define TAGCD_SECTORSIZE 0x0004
#define TAGCD_XLECC 0x0005
#define TAGCD_EJECTRESET 0x0006

/* Information
 *
 * Information/Status structure describes current speed settings
 * for read and play commands, sector size, audio attenuation
 * precision, and drive status.
 */

struct CDInfo {
    /* Default */
    WORD PlaySpeed; /* Audio play speed (75) */
    WORD ReadSpeed; /* Data-rate of CD READ command (Max) */
    WORD ReadLSpeed; /* Data-rate of CD READ command (Max) (75) */
    WORD SectorSize; /* Number of bytes per sector (2048) */
    WORD XLECC; /* CDXL ECC enabled/disabled */
    WORD EjectReset; /* Reset on eject enabled/disabled */
    WORD Reserved[4]; /* Reserved for future expansion */

    WORD MaxSpeed; /* Maximum speed drive can handle (75, 150) */
    WORD AudioPrecision; /* 0 = no attenuator, 1 = mute only. */
    WORD Status; /* other = (# levels - 1) */
    WORD Reserved2[4]; /* See flags below */
};

```

devices/cd.h

```

/* Flags for Status */
#define CDSTSB_CLOSED 0 /* Drive door is closed */
#define CDSTSB_DISK 1 /* A disk has been detected */
#define CDSTSB_SPIN 2 /* Disk is spinning (motor is on) */
#define CDSTSB_TOC 3 /* Table of contents read. Disk is valid. */
#define CDSTSB_CDROM 4 /* Track 1 contains CD-ROM data */
#define CDSTSB_PLAYING 5 /* Audio is playing */
#define CDSTSB_PAUSED 6 /* Pause mode (pauses on play command) */
#define CDSTSB_SEARCH 7 /* Search mode (Fast Forward/Fast Reverse) */
#define CDSTSB_DIRECTION 8 /* Search direction (0 = Forward, 1 = Reverse) */

#define CDSTSF_CLOSED 0x0001
#define CDSTSF_DISK 0x0002
#define CDSTSF_SPIN 0x0004
#define CDSTSF_TOC 0x0008
#define CDSTSF_CDROM 0x0010
#define CDSTSF_PLAYING 0x0020
#define CDSTSF_PAUSED 0x0040
#define CDSTSF_SEARCH 0x0080
#define CDSTSF_DIRECTION 0x0100

/* Nodes for CD_SEARCH */
#define CDMODE_NORMAL 0 /* Normal play at current play speed */
#define CDMODE_FWD 1 /* Fast forward play (skip-play forward) */
#define CDMODE_REV 2 /* Fast reverse play (skip-play reverse) */

/* Position Information
 *
 * Position information can be described in two forms: MSF and LSN
 * form. MSF (Minutes, Seconds, Frames) form is a time encoding.
 * LSN (Logical Sector Number) form is frame (sector) count.
 * The desired form is selected using the io_Flags field of the
 * io_Scdcd structure. The flags and the union are described
 * below.
 */

struct RMSF {
    BYTE Reserved; /* Reserved (always zero) */
    BYTE Minute; /* Minutes (0-72ish) */
    BYTE Second; /* Seconds (0-59) */
    BYTE Frame; /* Frame (0-74) */
};

union LSNMSF {
    struct RMSF MSF; /* Minute, Second, Frame */
    ULONG LSN; /* Logical Sector Number */
};

/* CD Transfer Lists
 *
 * A CDXL node is a double link node; however only single linkage
 * is used by the device driver. If you wish to construct a
 * transfer list manually, it is only necessary to define the
 * min_succ pointer of the MinNode. You may also use the Exec
 * list functions by defining a list or MinList structure and by
 * using the AddHead/AddTail functions to create the list. This
 * will create a double-linked list. Although a double-linked
 * list is not required by the device driver, you may wish use it
 * for your own purposes. Don't forget to initialize the
 * the List/MinList before using it!
 */

```


devices/cd.h

```

*****
struct CDXL {
    char MinNode;
    char Node;
    LONG *Buffer;
    LONG Length;
    LONG Actual;
    APTR InData;
    VOID (*Intcode)();
};

/* double linkage
 * data destination (word aligned)
 * must be even # bytes
 * bytes transferred
 * interrupt server data segment
 * interrupt server code entry
 */

/*****
 * CD Table of Contents
 *
 * The CD_TOC command returns an array of CDToc entries.
 * Entry zero contains summary information describing how many
 * tracks the disk has and the play-time of the disk.
 * Entries 1 through N (N = Number of tracks on disk) contain
 * information about the track.
 *
 *****/

struct TOCsummary {
    UBYTE FirstTrack; /* First track on disk (always 1)
    UBYTE LastTrack; /* Last track on disk
    union LSNMSF LeadOut; /* Beginning of lead-out track (end of disk)
};

struct TOCentry {
    UBYTE CtlAddr; /* Q-Code info
    UBYTE Track; /* Track number
    union LSNMSF Position; /* Start position of this track
};

union CDToc {
    struct TOCsummary Summary; /* First entry (0) is summary information
    struct TOCentry Entry; /* Entries 1-N are track entries
};

/*****
 * Q-Code Packets
 *
 * Q-Code packets are only returned when audio is playing.
 * Currently, only position packets are returned (ADR_POSITION)
 * The other ADR_ types are almost never encoded on the disk
 * and are of little use anyway. To avoid making the QCode
 * structure a union, these other ADR_ structures are not defined.
 *
 *****/

struct QCode {
    UBYTE CtlAddr; /* Data type / QCode type
    UBYTE Track; /* Track number
    UBYTE Index; /* Track subindex number
    UBYTE Zero; /* The "zero" byte of Q-Code packet
    union LSNMSF TrackPosition; /* Position from start of track
    union LSNMSF DiskPosition; /* Position from start of disk
};

#define CTLADDR_CTLMASK 0xF0 /* Control field

```

devices/cd.h

```

#define CTL_CTLMASK 0xD0 /* To be ANDed with CtlAddr before compared
#define CTL_2AUD 0x00 /* 2 audio channels without preemphasis
#define CTL_4AUD 0x10 /* 2 audio channels with preemphasis
#define CTL_4AUD 0x80 /* 4 audio channels without preemphasis
#define CTL_4AUDEMPH 0x90 /* 4 audio channels with preemphasis
#define CTL_DATA 0x40 /* CD-ROM Data
#define CTL_COPYMASK 0x20 /* To be ANDed with CtlAddr before compared
#define CTL_COPY 0x20 /* When true, this audio/data can be copied
#define CTLADDR_ADRMASK 0x0F /* Address field
#define ADR_POSITION 0x01 /* Q-Code is position information
#define ADR_UPC 0x02 /* Q-Code is UPC information (not used)
#define ADR_ISRC 0x03 /* Q-Code is ISRC (not used)
#define ADR_HYBRID 0x05 /* This disk is a hybrid disk
#endif

```

devices/cd.i

```

IFND DEVICES_CD_I
DEVICES_CD_I SET _ 1
**
** $VER: cd.i 1.12 (12.8.93)
** Includes Release 40.13
**
** cd.device include file
**
** (C) Copyright 1992-1993 Commodore-Amiga, Inc.
** All Rights Reserved
**
** INCLUDE "exec/types.i"
** INCLUDE "exec/nodes.i"
**
** *****
** * CD Commands
** * *****
**
CD_RESET equ 1
CD_READ equ 2
CD_WRITE equ 3
CD_UPDATE equ 4
CD_CLEAR equ 5
CD_STOP equ 6
CD_START equ 7
CD_FLUSH equ 8
CD_MOTOR equ 9
CD_SEEK equ 10
CD_FORMAT equ 11
CD_REMOVE equ 12
CD_CHANGENUM equ 13
CD_CHANGESECT equ 14
CD_PROTSTATUS equ 15

CD_GETDRIVETYPE equ 18
CD_GETNUMTRACKS equ 19
CD_ADCHANGEINT equ 20
CD_RECHANGEINT equ 21
CD_GETGEOMETRY equ 22
CD_EJECT equ 23

CD_INFO equ 32
CD_CONFIG equ 33
CD_TCHSF equ 34
CD_FOCLSN equ 35
CD_READXL equ 36
CD_PLAYTRACK equ 37
CD_PLAYMSF equ 38
CD_PLAYLSN equ 39
CD_PAUSE equ 40
CD_SEARCH equ 41
CD_OCODEMSF equ 42
CD_OCODELSN equ 43
CD_ATTENUATE equ 44
CD_ADDFRAMEINT equ 45
CD_REMFRAMEINT equ 46

** *****
** * Device Driver Error Codes
** * *****
**
CDERR_OPENFAIL equ -1 ; device/unit failed to open
CDERR_ABORTED equ -2 ; request terminated early

```

devices/cd.i

```

CDERR_NOCMD equ -3 ; command not supported by device
CDERR_BADLENGTH equ -4 ; invalid length (IO LENGTH/IO OFFSET)
CDERR_BADADDRESS equ -5 ; invalid address (IO DATA misaligned)
CDERR_UNITBUSY equ -6 ; device opens ok, but unit is busy
CDERR_SELFTEST equ -7 ; hardware failed self-test

CDERR_NotSpecified equ 20 ; general catchall
CDERR_NoSecHdr equ 21 ; couldn't even find a sector
CDERR_BadSecPreamble equ 22 ; sector looked wrong
CDERR_BadSecID equ 23 ; ditto
CDERR_BadHdrSum equ 24 ; header had incorrect checksum
CDERR_BadSecSum equ 25 ; data had incorrect checksum
CDERR_TooFwSecs equ 26 ; couldn't find enough sectors
CDERR_BadSecHdr equ 27 ; another "sector looked wrong"
CDERR_Writeprot equ 28 ; can't write to a protected disk
CDERR_NoDisk equ 29 ; no disk in the drive
CDERR_NoDisk equ 30 ; couldn't find track 0
CDERR_NoMem equ 31 ; ran out of memory
CDERR_BadUnitNum equ 32 ; asked for a unit > NUMUNITS
CDERR_BadDriveType equ 33 ; not a drive cd.device understands
CDERR_DriveUse equ 34 ; someone else allocated the drive
CDERR_PostReset equ 35 ; user hit reset; awaiting doom
CDERR_BadDataTypes equ 36 ; data on disk is wrong type
CDERR_InvalidState equ 37 ; invalid cmd under current conditions

CDERR_Phase equ 42 ; illegal or unexpected SCSI phase
CDERR_NoBoard equ 50 ; open failed for non-existent board

** *****
** * Configuration
** * *****
**
** The drive is configured by TagList items defined as follows:
** *****
**
TAGCD_PLAYSPD equ $0001
TAGCD_READSPD equ $0002
TAGCD_READXLSPD equ $0003
TAGCD_SECTORSIZE equ $0004
TAGCD_XLECC equ $0005
TAGCD_EJECTRESET equ $0006

** *****
** * Information
** * *****
**
** Information/Status structure describes current speed settings
** for read and play commands, sector size, audio attenuation
** precision, and drive status.
** *****
**
STRUCTURE CDINFO_0
** *****
**
UNWORD CDINFO_PlaySpeed ; Audio play speed
UNWORD CDINFO_ReadSpeed ; Data-rate of CD READ command (75)
UNWORD CDINFO_ReadXLSPD ; Data-rate of CD-READXL command (Max)
UNWORD CDINFO_SectorSize ; Number of bytes-per sector (75)
UNWORD CDINFO_XLECC ; CDXL ECC enabled/disabled (2048)
UNWORD CDINFO_EjectReset ; Reset on eject enabled/disabled
STRUCT CDINFO_Reserve1,8 ; Reserved for future expansion

UNWORD CDINFO_MaxSpeed ; Maximum speed drive can handle (75, 150)
UNWORD CDINFO_AudioPrecision ; 0 = no attenuator, 1 = mute only,
; other = (# levels - 1)
UNWORD CDINFO_Status ; See flags below
STRUCT CDINFO_Reserve2,8 ; Reserved for future expansion
LABEL CDINFO_SIZE

```


devices/cd.i

```

CTL_2AUDEMPH    equ $10    ; 2 audio channels with preemphasis
CTL_4AUD        equ $80    ; 4 audio channels without preemphasis
CTL_4AUDEMPH    equ $90    ; 4 audio channels with preemphasis
CTL_DATA        equ $40    ; CD-ROM Data
CTL_COPYMASK    equ $20    ; To be ANDed with CtlAdr before compared
CTL_COPY        equ $20    ; When true, this audio/data can be copied
CTLADR_ADRMASK  equ $0F    ; Address field
ADR_POSITION    equ $01    ; Q-Code is position information
ADR_UPC        equ $02    ; Q-Code is UPC information (not used)
ADR_ISRC       equ $03    ; Q-Code is ISRC (not used)
ADR_HYBRID     equ $05    ; This disk is a hybrid disk

```

ENDC

libraries/lowlevel.h

```

/*
*****
$VER: lowlevel.h 40.6 (30.7.93)
includes Release 40.13

lowlevel.library interface structures and definitions.

(C) Copyright 1993 Commodore-Amiga, Inc.
All Rights Reserved
*/
*****

#ifndef EXEC_TYPES_H
#include <exec/types.h>
#endif

#ifndef UTILITY_TAGITEM_H
#include <utility/tagitem.h>
#endif
*****

/* structure for use with QueryKeys() */
struct KeyQuery
{
    UWORD kq_KeyCode;
    BOOL kq_Pressed;
};
*****

/* Bits in the return value of GetKey() */
#define LLKB_LSHIFT      16
#define LLKB_RSHIFT     17
#define LLKB_CAPSLOCK   18
#define LLKB_CTRL       19
#define LLKB_ALT        20
#define LLKB_LALT       21
#define LLKB_RAMIGA     22
#define LLKB_RAMIGA     23
#define LLKF_LSHIFT     (1<<LLKB_LSHIFT)
#define LLKF_RSHIFT     (1<<LLKB_RSHIFT)
#define LLKF_CAPSLOCK   (1<<LLKB_CAPSLOCK)
#define LLKF_CTRL       (1<<LLKB_CTRL)
#define LLKF_LALT       (1<<LLKB_LALT)
#define LLKF_RAMIGA     (1<<LLKB_RAMIGA)
#define LLKF_RAMIGA     (1<<LLKB_RAMIGA)
*****

/* Tags for SetJoyPortAttrs() */
#define SJA_Dummy      (TAG_USER+0xc00100)
#define SJA_Reinitiaize (SJA_Dummy+1) /* force type to mouse, joy, game cntrlr */
#define SJA_TypeTag     (SJA_Dummy+2) /* free potdjo bits, reset to autosense */
*****

/* Controller types for SJA_Type tag */
#define SJA_TYPE_AUTOSENSE 0
#define SJA_TYPE_GAMECTR 1
#define SJA_TYPE_MOUSE 2
#define SJA_TYPE_JOYSTK 3

```

```

/* ***** */
/* ReadJoyPort() return value definitions */

/* Port types */
#define JP_TYPE_NOTAVAIL (00<<28) /* port data unavailable */
#define JP_TYPE_GAMECTLR (01<<28) /* port has game controller */
#define JP_TYPE_MOUSE (02<<28) /* port has mouse */
#define JP_TYPE_JOYSTK (03<<28) /* port has joystick */
#define JP_TYPE_UNKNOWN (04<<28) /* port has unknown device */
#define JP_TYPE_MASK (15<<28) /* controller type */

/* Button types, valid for all types except JP_TYPE_NOTAVAIL */
#define JPB_BUTTON_BLUE 22 /* Blue - stop; Right Mouse */
#define JPB_BUTTON_RED 23 /* Red - Select; Left Mouse; Joystick Fire */
#define JPB_BUTTON_YELLOW 20 /* Yellow - Repeat */
#define JPB_BUTTON_GREEN 21 /* Green - Shuffle */
#define JPB_BUTTON_FORWARD 19 /* Charcoal - Forward */
#define JPB_BUTTON_REVERSE 18 /* Charcoal - Reverse */
#define JPB_BUTTON_PLAY 17 /* Grey - Play/Pause; Middle Mouse */
#define JPB_BUTTON_BLUE2 (1<<JPB_BUTTON_BLUE)
#define JPB_BUTTON_RED2 (1<<JPB_BUTTON_RED)
#define JPB_BUTTON_YELLOW2 (1<<JPB_BUTTON_YELLOW)
#define JPB_BUTTON_GREEN2 (1<<JPB_BUTTON_GREEN)
#define JPB_BUTTON_FORWARD2 (1<<JPB_BUTTON_FORWARD)
#define JPB_BUTTON_REVERSE2 (1<<JPB_BUTTON_REVERSE)
#define JPB_BUTTON_PLAY2 (1<<JPB_BUTTON_PLAY)
#define JPB_BUTTON_BLUE1JPB_BUTTON_RED1JPB_BUTTON_YELLOW1JPB_BUTTON_GREEN1JPB_BUTTON_FORWARD1JPB_BUTTON_REVERSE1JPB_BUTTON_PLAY1JPB_BUTTON_REVERSE1JPB_BUTTON_PLAY1)

/* Direction types, valid for JP_TYPE_GAMECTLR and JP_TYPE_JOYSTK */
#define JPD_JOY_UP 3
#define JPD_JOY_DOWN 2
#define JPD_JOY_LEFT 1
#define JPD_JOY_RIGHT 0
#define JPD_JOY_UP2 (1<<JPB_JOY_UP)
#define JPD_JOY_DOWN2 (1<<JPB_JOY_DOWN)
#define JPD_JOY_LEFT2 (1<<JPB_JOY_LEFT)
#define JPD_JOY_RIGHT2 (1<<JPB_JOY_RIGHT)
#define JPD_DIRECTION_MASK (JPD_JOY_UP|JPD_JOY_DOWN|JPD_JOY_LEFT|JPD_JOY_RIGHT)

/* Mouse position reports, valid for JP_TYPE_MOUSE */
#define JPM_HOR2_MASK (255<<0) /* horizontal position */
#define JPM_VERT_MASK (255<<8) /* vertical position */
#define JPM_HOR2_MASK1JPM_VERT_MASK1JPM_HOR2_MASK1)

/* Obsolete ReadJoyPort() definitions, here for source code compatibility only.
 * Please do NOT use in new code.
 */
#define JPB_BTN1 JPB_BUTTON_BLUE
#define JPB_BTN1 JPB_BUTTON_BLUE
#define JPB_BTN2 JPB_BUTTON_RED
#define JPB_BTN2 JPB_BUTTON_RED
#define JPB_BTN3 JPB_BUTTON_YELLOW
#define JPB_BTN3 JPB_BUTTON_YELLOW
#define JPB_BTN4 JPB_BUTTON_GREEN
#define JPB_BTN4 JPB_BUTTON_GREEN
#define JPB_BTN5 JPB_BUTTON_FORWARD
#define JPB_BTN5 JPB_BUTTON_FORWARD
#define JPB_BTN6 JPB_BUTTON_REVERSE
#define JPB_BTN6 JPB_BUTTON_REVERSE
#define JPB_BTN7 JPB_BUTTON_PLAY
#define JPB_BTN7 JPB_BUTTON_PLAY
#define JPB_UP JPB_JOY_UP
#define JPB_UP JPB_JOY_UP
#define JPB_DOWN JPB_JOY_DOWN
#define JPB_DOWN JPB_JOY_DOWN
#define JPB_LEFT JPB_JOY_LEFT
#define JPB_LEFT JPB_JOY_LEFT
#define JPB_RIGHT JPB_JOY_RIGHT
#define JPB_RIGHT JPB_JOY_RIGHT

```


libraries/lowlevel.h

```

/*****
/* Tags for SystemControl() */
#define SCON_Dummy (TAG_USER+0x00000000)
#define SCON_TakeOverSys (SCON_Dummy+0)
#define SCON_KillReq (SCON_Dummy+1)
#define SCON_CDReboot (SCON_Dummy+2)
#define SCON_StopInput (SCON_Dummy+3)
#define SCON_AddCreateKeys (SCON_Dummy+4)
#define SCON_RemCreateKeys (SCON_Dummy+5)

/* Reboot control values for use with SCON_CDReboot tag */
#define CDReboot_On 1
#define CDReboot_Off 0
#define CDReboot_Default 2

/*****
/* Rawkey codes returned when using SCON_AddCreateKeys with SystemControl() */

#define RAWKEY_PORT0_BUTTON_BLUE 0x72
#define RAWKEY_PORT0_BUTTON_RED 0x78
#define RAWKEY_PORT0_BUTTON_YELLOW 0x77
#define RAWKEY_PORT0_BUTTON_GREEN 0x76
#define RAWKEY_PORT0_BUTTON_FORWARD 0x75
#define RAWKEY_PORT0_BUTTON_REVERSE 0x74
#define RAWKEY_PORT0_BUTTON_PLAY 0x73
#define RAWKEY_PORT0_JOY_UP 0x79
#define RAWKEY_PORT0_JOY_DOWN 0x7A
#define RAWKEY_PORT0_JOY_LEFT 0x7C
#define RAWKEY_PORT0_JOY_RIGHT 0x7B

#define RAWKEY_PORT1_BUTTON_BLUE 0x172
#define RAWKEY_PORT1_BUTTON_RED 0x178
#define RAWKEY_PORT1_BUTTON_YELLOW 0x177
#define RAWKEY_PORT1_BUTTON_GREEN 0x176
#define RAWKEY_PORT1_BUTTON_FORWARD 0x175
#define RAWKEY_PORT1_BUTTON_REVERSE 0x174
#define RAWKEY_PORT1_BUTTON_PLAY 0x173
#define RAWKEY_PORT1_JOY_UP 0x179
#define RAWKEY_PORT1_JOY_DOWN 0x17A
#define RAWKEY_PORT1_JOY_LEFT 0x17C
#define RAWKEY_PORT1_JOY_RIGHT 0x17B

#define RAWKEY_PORT2_BUTTON_BLUE 0x272
#define RAWKEY_PORT2_BUTTON_RED 0x278
#define RAWKEY_PORT2_BUTTON_YELLOW 0x277
#define RAWKEY_PORT2_BUTTON_GREEN 0x276
#define RAWKEY_PORT2_BUTTON_FORWARD 0x275
#define RAWKEY_PORT2_BUTTON_REVERSE 0x274
#define RAWKEY_PORT2_BUTTON_PLAY 0x273
#define RAWKEY_PORT2_JOY_UP 0x279
#define RAWKEY_PORT2_JOY_DOWN 0x27A
#define RAWKEY_PORT2_JOY_LEFT 0x27C
#define RAWKEY_PORT2_JOY_RIGHT 0x27B

#define RAWKEY_PORT3_BUTTON_BLUE 0x372
#define RAWKEY_PORT3_BUTTON_RED 0x378
#define RAWKEY_PORT3_BUTTON_YELLOW 0x377
#define RAWKEY_PORT3_BUTTON_GREEN 0x376
#define RAWKEY_PORT3_BUTTON_FORWARD 0x375
#define RAWKEY_PORT3_BUTTON_REVERSE 0x374
#define RAWKEY_PORT3_BUTTON_PLAY 0x373
#define RAWKEY_PORT3_JOY_UP 0x379
#define RAWKEY_PORT3_JOY_DOWN 0x37A
#define RAWKEY_PORT3_JOY_LEFT 0x37C
#define RAWKEY_PORT3_JOY_RIGHT 0x37B

/*****

```

libraries/lowlevel.h

```

/* Return values for GetLanguageSelection() */
#define LANG_UNKNOWN 0
#define LANG_AMERICAN 1 /* American English */
#define LANG_ENGLISH 2 /* British English */
#define LANG_GERMAN 3
#define LANG_FRENCH 4
#define LANG_SPANISH 5
#define LANG_ITALIAN 6
#define LANG_PORTUGUESE 7
#define LANG_DUTCH 8
#define LANG_NORWEGIAN 9
#define LANG_FINNISH 10
#define LANG_SWEDISH 11
#define LANG_JAPANESE 12
#define LANG_CHINESE 13
#define LANG_ARABIC 14
#define LANG_GREEK 15
#define LANG_HEBREW 16
#define LANG_KOREAN 17

/*****
#endif /* LIBRARIES_LOWLEVEL_H */

```


libraries/lowlevel.i

```

IFND LIBRARIES LOWLEVEL_I
LIBRARIES_LOWLEVEL_I SET 1
**
** $VER: lowlevel.i 40.7 (30.7.93)
** Includes Release 40.13
**
** lowlevel.library interface structures and definitions
**
** (C) Copyright 1993 Commodore-Amiga, Inc.
** All Rights Reserved
**
;-----
; EXEC TYPES I
INCLUDE "exec/types.i"
ENDC

; UTILITY TAGITEM I
INCLUDE "utility/tagitem.i"
ENDC

;-----
; structure for use with QueryKeys()
STRUCTURE KeyQuery 0
  UMWORD kq_KeyCode
  UMWORD kq_Pressed
  LABEL KeyQuery_SIZEOF
ENDC

;-----
; bits in the return value of GetKey()
BITDEF LK_LSHIFT 16
BITDEF LK_RSHIFT 17
BITDEF LK_CAPSLOCK 18
BITDEF LK_CTRL 19
BITDEF LK_LALT 20
BITDEF LK_RALT 21
BITDEF LK_LAMIGA 22
BITDEF LK_RAMIGA 23

;-----
; Tags for SetJoyPortAttrs()
SJA Dummy equ (TAG_USER+$00C00100)
SJA_Type equ (SJA_Dummy+1) ; force type to mouse, joy, game ctrlr
SJA_Initialize equ (SJA_Dummy+2) ; free portg bits, reset to autosense

; Controller types for SJA_Type tag
SJA_Type_AUTOSENSE equ 1
SJA_Type_GAMECTRL equ 2
SJA_Type_MOUSE equ 3

;-----
; ReadJoyPort() return value definitions
; Port types
JP_Type_NOTAVAIL equ (0<<28) ; port data unavailable
JP_Type_GAMECTRL equ (01<<28) ; port has game controller
JP_Type_MOUSE equ (02<<28) ; port has mouse
JP_Type_JOYSTICK equ (03<<28) ; port has joystick
JP_Type_UNKNOWN equ (04<<28) ; port has unknown device
JP_Type_MASK equ (15<<28) ; controller type

; Button types, valid for all types except JP_Type_NOTAVAIL
BITDEF JP_BUTTON_BLUE,23 ; Blue - Stop; Right Mouse
BITDEF JP_BUTTON_RED,22 ; Red - Select; Left Mouse; Joystick Fire
BITDEF JP_BUTTON_YELLOW,21 ; Yellow - Repeat

```

libraries/lowlevel.i

```

BITDEF JP_BUTTON_GREEN,20 ; Green - Shuffle
BITDEF JP_BUTTON_FORWARD,19 ; Charcoal - Forward
BITDEF JP_BUTTON_REVERSE,18 ; Charcoal - Reverse
BITDEF JP_BUTTON_PLAY,17 ; Gray - Play/Pause; Middle Mouse

JP_BUTTON_MASK equ (JP_BUTTON_BLUE|JP_BUTTON_RED|JP_BUTTON_YELLOW|
JP_BUTTON_GREEN|JP_BUTTON_FORWARD|
JP_BUTTON_REVERSE|JP_BUTTON_PLAY)

; Direction types, valid for JP_Type_GAMECTRL and JP_Type_JOYSTICK
BITDEF JP_JOY_UP,3
BITDEF JP_JOY_DOWN,2
BITDEF JP_JOY_LEFT,1
BITDEF JP_JOY_RIGHT,0

JP_DIRECTION_MASK equ (JP_JOY_UP|JP_JOY_DOWN|JP_JOY_LEFT|JP_JOY_RIGHT)

; Mouse position reports, valid for JP_Type_MOUSE
JP_MOUSE_MASK equ (255<<0) ; horizontal position
JP_MOUSE_MASK equ (255<<8) ; vertical position
JP_MOUSE_MASK equ (JP_MOUSE_MASK|JP_AVERT_MASK)

; Obsolete ReadJoyPort() definitions, here for source code compatibility only.
; Please do NOT use in new code.
BITDEF JP_BTN1_JPB_BUTTON_BLUE
BITDEF JP_BTN2_JPB_BUTTON_RED
BITDEF JP_BTN3_JPB_BUTTON_YELLOW
BITDEF JP_BTN4_JPB_BUTTON_GREEN
BITDEF JP_BTN5_JPB_BUTTON_FORWARD
BITDEF JP_BTN6_JPB_BUTTON_REVERSE
BITDEF JP_BTN7_JPB_BUTTON_PLAY
BITDEF JP_UP_JPB_JOY_UP
BITDEF JP_DOWN_JPB_JOY_DOWN
BITDEF JP_LEFT_JPB_JOY_LEFT
BITDEF JP_RIGHT_JPB_JOY_RIGHT

;-----
; Tags for SystemControl()
SCON_Dummy equ (TAG_USER+$00C00000)
SCON_TimeoverSys equ (SCON_Dummy+0)
SCON_AllInReq equ (SCON_Dummy+1)
SCON_CDReboot equ (SCON_Dummy+2)
SCON_StopInput equ (SCON_Dummy+3)
SCON_AddcreateKeys equ (SCON_Dummy+4)
SCON_RemcreateKeys equ (SCON_Dummy+5)

; Reboot control values for use with the SCON_CDReboot tag
CDReboot_On equ 1
CDReboot_Off equ 0
CDReboot_Default equ 2

;-----
; Rawkey codes returned when using SCON_AddcreateKeys with SystemControl()
RAWKEY_PORT0_BUTTON_BLUE equ $72
RAWKEY_PORT0_BUTTON_RED equ $78
RAWKEY_PORT0_BUTTON_YELLOW equ $77
RAWKEY_PORT0_BUTTON_GREEN equ $76
RAWKEY_PORT0_BUTTON_FORWARD equ $75
RAWKEY_PORT0_BUTTON_REVERSE equ $74
RAWKEY_PORT0_BUTTON_PLAY equ $73
RAWKEY_PORT0_JOY_UP equ $79
RAWKEY_PORT0_JOY_DOWN equ $7A
RAWKEY_PORT0_JOY_LEFT equ $7C
RAWKEY_PORT0_JOY_RIGHT equ $7B

RAWKEY_PORT1_BUTTON_BLUE equ $172
RAWKEY_PORT1_BUTTON_RED equ $178
RAWKEY_PORT1_BUTTON_YELLOW equ $177
RAWKEY_PORT1_BUTTON_GREEN equ $176

```

libraries/lowlevel.i

```

RANKY_PORT1_BUTTON_FORWARD equ $175
RANKY_PORT1_BUTTON_REVERSE equ $174
RANKY_PORT1_BUTTON_PLAY equ $173
RANKY_PORT1_JOY_UP equ $179
RANKY_PORT1_JOY_DOWN equ $17A
RANKY_PORT1_JOY_LEFT equ $17C
RANKY_PORT1_JOY_RIGHT equ $17B

RANKY_PORT2_BUTTON_BLUE equ $272
RANKY_PORT2_BUTTON_RED equ $278
RANKY_PORT2_BUTTON_YELLOW equ $277
RANKY_PORT2_BUTTON_GREEN equ $276
RANKY_PORT2_BUTTON_FORWARD equ $275
RANKY_PORT2_BUTTON_REVERSE equ $274
RANKY_PORT2_BUTTON_PLAY equ $273
RANKY_PORT2_JOY_UP equ $279
RANKY_PORT2_JOY_DOWN equ $27A
RANKY_PORT2_JOY_LEFT equ $27C
RANKY_PORT2_JOY_RIGHT equ $27B

RANKY_PORT3_BUTTON_BLUE equ $372
RANKY_PORT3_BUTTON_RED equ $378
RANKY_PORT3_BUTTON_YELLOW equ $377
RANKY_PORT3_BUTTON_GREEN equ $376
RANKY_PORT3_BUTTON_FORWARD equ $375
RANKY_PORT3_BUTTON_REVERSE equ $374
RANKY_PORT3_BUTTON_PLAY equ $373
RANKY_PORT3_JOY_UP equ $379
RANKY_PORT3_JOY_DOWN equ $37A
RANKY_PORT3_JOY_LEFT equ $37C
RANKY_PORT3_JOY_RIGHT equ $37B

```

```

; return value for GetLanguageSelection()
LANG_UNKNOWN equ 0
LANG_AMERICAN equ 1 ; American English
LANG_ENGLISH equ 2 ; British English
LANG_GERMAN equ 3
LANG_FRENCH equ 4
LANG_SPANISH equ 5
LANG_ITALIAN equ 6
LANG_PORTUGUESE equ 7
LANG_DANISH equ 8
LANG_DUTCH equ 9
LANG_NORWEGIAN equ 10
LANG_FINNISH equ 11
LANG_SWEDISH equ 12
LANG_JAPANESE equ 13
LANG_CHINESE equ 14
LANG_ARABIC equ 15
LANG_GREEK equ 16
LANG_HEBREW equ 17
LANG_KOREAN equ 18

```

ENDC ; LIBRARIES_LOWLEVEL_I

libraries/nonvolatile.h

```

#ifndef LIBRARIES_NONVOLATILE_H
#define LIBRARIES_NONVOLATILE_H

/*
** $VER: nonvolatile.h 40.0 (30.7.93)
** Includes Release 40.13
** nonvolatile.library interface structures and definitions.
** (C) Copyright 1992-1993 Commodore-Amiga, Inc.
** All Rights Reserved
** */

/*****
**
** #ifndef EXEC_TYPES_H
** #include <exec/types.h>
** #endif
**
** #ifndef EXEC_NODES_H
** #include <exec/nodes.h>
** #endif
**
** */
/*****
**
** struct NVInfo
** {
**     ULONG nvi_MaxStorage;
**     ULONG nvi_FreeStorage;
** };
**
** */
/*****
**
** struct NVEntry
** {
**     struct MinNode nve_Node;
**     STRPTR nve_Name;
**     ULONG nve_Size;
**     ULONG nve_Protection;
** };
**
** /* bit definitions for mask in SetNVProtection(). Also used for
**    * NVEntry.nve_Protection.
**    */
** #define NVEB_DELETE 0
** #define NVEB_APPNAME 31
**
** #define NVEF_DELETE (1<<NVEB_DELETE)
** #define NVEF_APPNAME (1<<NVEB_APPNAME)
**
** */
/*****
**
** /* errors from StoreNV() */
** #define NVERR_BADNAME 1
** #define NVERR_WRITEPROT 2
** #define NVERR_FAIL 3
** #define NVERR_FATAL 4
**
** */
/*****
**
** /* determine the size of data returned by this library */
** #define SizenVData(DataPtr) (((ULONG *) DataPtr)[-1]) - 4)
**
** */
/*****
**
** #endif /* LIBRARIES_NONVOLATILE_H */

```

libraries/nonvolatile.i

```

IFND LIBRARIES_NONVOLATILE_I
LIBRARIES_NONVOLATILE_I SET 1
**
** $VER: nonvolatile.i 40.10 (3.8.93)
** Includes Release 40.13
** nonvolatile.library interface structures and definitions.
** (C) Copyright 1992-1993 Commodore-Amiga, Inc.
** All Rights Reserved
**
** */
/*****
**
** IFND EXEC_TYPES_I
** #include "exec/types.i"
** ENDC
**
** IFND EXEC_NODES_I
** #include "exec/nodes.i"
** ENDC
**
** */
/*****
**
** STRUCTURE NVInfo_0
** ULONG nvi_MaxStorage
** ULONG nvi_FreeStorage
** LABEL NVINFO_SIZE
**
** */
/*****
**
** STRUCTURE NVEntry_0
** STRUCT nve_Node,MLN_SIZE
**     STRPTR nve_Name
**     ULONG nve_Size
**     ULONG nve_Protection
** LABEL NVENTRY_SIZE
**
** */
/*****
**
** ; Bit definitions for mask in SetNVProtection(). Also used for
** ; NVEntry.nve_Protection.
**
**     BITDEF NVE_DELETE,0
**     BITDEF NVE_APPNAME,31
**
** */
/*****
**
** ; errors from StoreNV()
** NVERR_BADNAME equ 1
** NVERR_WRITEPROT equ 2
** NVERR_FAIL equ 3
** NVERR_FATAL equ 4
**
** */
/*****
**
** ; determines the size of data returned by this library
** SizenVData
**     MACRO :DataPtr SizeReg
**         move.l -4(1),/2
**         subq.l #4,/2
**     ENDM
**
** */
/*****
**
** ENDC ; LIBRARIES_NONVOLATILE_I

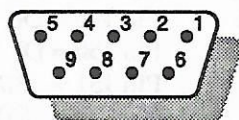
```


5

Expansion Connectors

Connectors And Expansion Port

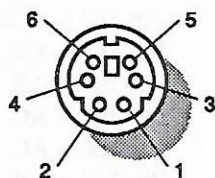
The joystick connectors have the same pinouts as the Amiga 500:



Joystick Port (DB9 male)

Pin 1 = Mouse V	Pin 6 = Mouse Button 1
Pin 2 = Mouse H	Pin 7 = +5v(100mA Max)
Pin 3 = Mouse VQ	Pin 8 = Ground
Pin 4 = Mouse HQ	Pin 9 = Mouse Button 3 (POTY)
Pin 5 = Mouse Button 2 (POTX)	

The auxiliary connector is for an optional keyboard. The pinouts are as follows:



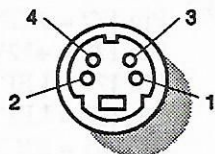
Auxiliary Port (6-pin female mini DIN, PS/2-type)

Pin 1 = Data	Pin 4 = +5 VDC (100 mA)
Pin 2 = n/c	Pin 5 = Clock
Pin 3 = Ground	Pin 6 = n/c

The Line Audio Output RCA jacks produce 2VRMS into an output impedance of 10KW.

The composite video output RCA jack produces a 1VP-P signal, coming close to meeting the RS170A specification for NTSC and the CCIR rep 624-2 specification for PAL.

The S-video connector provides 1VP-P luma and 320mVP-P chroma video signals. These signals are compatible with S-video monitors and C64-type monitors with separate chroma/luma inputs.

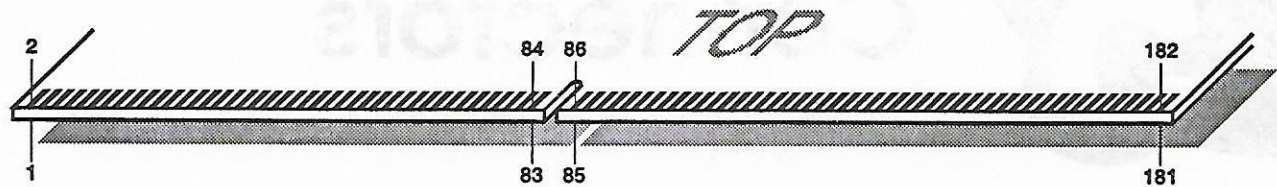


S-Video Port (4-pin female mini DIN)

Pin 1 = Ground	Pin 3 = 1VP-P Luma
Pin 2 = Ground	Pin 4 = 320m VP-P Chroma

The RF modulator output RCA jack provides an RF modulated version of the composite video and monaural audio on Channel 3 or 4 (VHF) for NTSC and Channel 36 (UHF) for PAL systems.

The expansion connector has gold fingers (15min plating min.) with 0.050" pitch. It will mate with a 32-bit Micro Channel Adapter (MCA) connector having 182 pins (AMP #650092-1). Note that while an MCA connector is used this is not an MCA expansion bus. The expansion connector is discussed later in this document.



Expansion Port (MCA TYPE 182 pin Card Edge, P1)

Pin 1 = A31	Pin 38 = A0	Pin 75 = D3	Pin 112 = _EXP_BG	Pin 147 = DR
Pin 2 = A30	Pin 39 = DGND	Pin 76 = D2	Pin 113 = _CPU_BG	Pin 148 = DG
Pin 3 = A29	Pin 40 = VCC	Pin 77 = D1	Pin 114 = _EXP_BR	Pin 149 = DB
Pin 4 = A28	Pin 41 = D31	Pin 78 = D0	Pin 115 = NC	Pin 150 = DI
Pin 5 = A27	Pin 42 = D30	Pin 79 = DGND	Pin 116 = NC	Pin 151 = _PIXELSW_EXT
Pin 6 = A26	Pin 43 = D29	Pin 80 = VCC	Pin 117 = _PUNT	Pin 152 = _PIXELSW
Pin 7 = A25	Pin 44 = D28	Pin 81 = _IPL2	Pin 118 = _RESET	Pin 153 = _BLANK
Pin 8 = A24	Pin 45 = D27	Pin 82 = _IPL1	Pin 119 = _INT2	Pin 154 = _PIXELCLK
Pin 9 = DGND	Pin 46 = D26	Pin 83 = _IPL0	Pin 120 = _INT6	Pin 155 = DGND
Pin 10 = VCC	Pin 47 = D25	Pin 84 = NC	Pin 121 = _KB_CLOCK	Pin 156 = VCC
Pin 11 = A23	Pin 48 = D24	Pin 85 = _RST	Pin 122 = _KB_DATA	Pin 157 = _CSYNC
Pin 12 = A22	Pin 49 = DGND	Pin 86 = _HLT	Pin 123 = _FIRE0	Pin 158 = CCK_B
Pin 13 = A21	Pin 50 = VCC	Pin 87 = NC	Pin 124 = _FIRE1	Pin 159 = _HSYNC
Pin 14 = A20	Pin 51 = D23	Pin 88 = NC	Pin 125 = _LED	Pin 160 = _VSYNC
Pin 15 = A19	Pin 52 = D22	Pin 89 = SIZE_1	Pin 126 = _ACTIVE	Pin 161 = VGND
Pin 16 = A18	Pin 53 = D21	Pin 90 = SIZE_0	Pin 127 = _RxD	Pin 162 = VGND
Pin 17 = A17	Pin 54 = D20	Pin 91 = _AS	Pin 128 = _TxD	Pin 163 = AR_EXT
Pin 18 = A16	Pin 55 = D19	Pin 92 = _DS	Pin 129 = _DKRD	Pin 164 = AR
Pin 19 = DGND	Pin 56 = D18	Pin 93 = R_W	Pin 130 = _DKWD	Pin 165 = AG_EXT
Pin 20 = VCC	Pin 57 = D17	Pin 94 = _BEER	Pin 131 = DGND	Pin 166 = AG
Pin 21 = A15	Pin 58 = D16	Pin 95 = NC	(SYSTEM)	Pin 167 = AB_EXT
Pin 22 = A14	Pin 59 = DGND	Pin 96 = _AVEC	Pin 132 = DKWE	Pin 168 = AB
Pin 23 = A13	Pin 60 = VCC	Pin 97 = _DSACK_1	Pin 133 = DGND	Pin 169 = VGND
Pin 24 = A12	Pin 61 = D15	Pin 98 = _DSACK_0	(_CONFIG_OUT)	Pin 170 = VGND
Pin 25 = A11	Pin 62 = D14	Pin 99 = CPUCLK_A	Pin 134 = NC	Pin 171 = _NTSC
Pin 26 = A10	Pin 63 = D13	Pin 100 = NC	Pin 135 = DGND	Pin 172 = _XCLKEN
Pin 27 = A9	Pin 64 = D12	Pin 101 = DGND	Pin 136 = +12V	Pin 173 = XCLK
Pin 28 = A8	Pin 65 = D11	Pin 102 = VCC	Pin 137 = DGND	Pin 174 = _EXT_VIDEO
Pin 29 = DGND	Pin 66 = D10	Pin 103 = FC2	Pin 138 = +12V	Pin 175 = DGND
Pin 30 = VCC	Pin 67 = D9	Pin 104 = FC1	Pin 139 = 17MHz	Pin 176 = VCC
Pin 31 = A7	Pin 68 = D8	Pin 105 = FC0	Pin 140 = _EXT_AUDIO	Pin 177 = AGND
Pin 32 = A6	Pin 69 = DGND	Pin 106 = NC	Pin 141 = DA_DATA	Pin 178 = +12V
Pin 33 = A5	Pin 70 = VCC	Pin 107 = NC	Pin 142 = _MUTE	Pin 179 = LEFT_EXT
Pin 34 = A4	Pin 71 = D7	Pin 108 = NC	Pin 143 = DA_LRCLK	Pin 180 = LEFT
Pin 35 = A3	Pin 72 = D6	Pin 109 = NC	Pin 144 = DA_BCLK	Pin 181 = RIGHT_EXT
Pin 36 = A2	Pin 73 = D5	Pin 110 = NC	Pin 145 = DGND	Pin 182 = RIGHT
Pin 37 = A1	Pin 74 = D4	Pin 111 = _CPU_BR	Pin 146 = VCC	

■ CD-ROM Subsystem

A separate PCB (2 layer) contains the CD electronics. This board contains the RF amp and servo chip, the audio DSP chip, and a small microcontroller for controlling the servo, focus, and communicating with the main CPU. The main CPU and RAM perform any prefetching, buffering and ECC as necessary.

The mechanism for the AmigaCD³² is a Sony KSM-2101BAM. The KSM-2101BAM has no drawer, or drawer motor. It is simply a spindle motor, traverse motor, and optical pickup. Chinon will provide the drive PCB. The electronics are based on the Sony CD chipset, which support double-speed operation. The microcontroller firmware has changed slightly in this version of the drive to (1) remove the firmware to control the drawer, and (2) revise the communications protocol to use a new 3-wire channel instead of the older 6-wire channel (to save pins on the ASIC) and (3) support multi-session discs (which is useful for PhotoCD support).

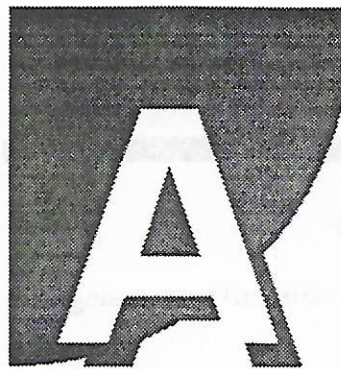
The software driver for the CD subsystem has been converted to run entirely on the 68EC020. The software also supports Akiko and its new DMA. This driver incorporates all the prefetching tricks and double-speed support completely from the 68EC020. It also supports both Mode 1 and Mode 2 (Form 1 and 2) discs. This is required to read all types of CD media called out in the Philips Yellow Book. This includes CD-I discs, PhotoCD[®] discs, and MPEG discs.

8 CD-ROM Subsystem

A CD-ROM (Compact Disc Read-Only Memory) is a type of optical storage medium. It is a small, round, reflective disc that is read by a laser beam. The data is stored in a spiral track on the surface of the disc. The disc is read by a CD-ROM drive, which is a device that can read the data from the disc and transfer it to the computer's memory.

The CD-ROM drive is a device that can read the data from the disc and transfer it to the computer's memory. It is a device that is connected to the computer's system unit. The drive is read by a laser beam, which is a device that can read the data from the disc and transfer it to the computer's memory. The drive is read by a laser beam, which is a device that can read the data from the disc and transfer it to the computer's memory. The drive is read by a laser beam, which is a device that can read the data from the disc and transfer it to the computer's memory.

The CD-ROM drive is a device that can read the data from the disc and transfer it to the computer's memory. It is a device that is connected to the computer's system unit. The drive is read by a laser beam, which is a device that can read the data from the disc and transfer it to the computer's memory. The drive is read by a laser beam, which is a device that can read the data from the disc and transfer it to the computer's memory. The drive is read by a laser beam, which is a device that can read the data from the disc and transfer it to the computer's memory.



Questions and Answers

Q1: *What should be in the startup-sequence? Just the name of the game executable or is something else needed?*

See Chapter 3 of this document for suggestions on startup-sequences.

Q2: *What non-volatile memory will exist on Amiga CD? Battery-backed RAM?*

The system supports both battery-backed RAM and floppies as non-volatile memory. Although the base AmigaCD³² unit has no floppy drives, you should not assume this will always be the case. Specifically, you should only call `nonvolatile.library` from a Process since a floppy drive may be involved on expanded systems.

► *Any indication of size?*

The system call `GetNVInfo()` in the `nonvolatile.library` can tell you exactly how much memory there is and how much is currently available. Be aware that the user may be playing multiple games one right after the other. Commodore recommends that you use no more than 25% of the total NV memory so that the system can remember the state of at least 4 different games while power is off.

Q3: *After calling `SystemControl()`, exactly what pieces of the OS are left in?*

Taking over the system with `SystemControl()` and the `SCON_TakeOverSys` tag simply shuts down Exec multitasking. This means you get the CPU all to yourself and any system tasks are effectively starved until you make a call that directly or indirectly calls `Wait()`, `WaitPort()`. All "pieces of the OS are left in," they just don't get any time unless you allow it.

Q4: *Which of the following (see list below) still operate under OS control and which can we safely take over and hit directly?*

Even with multitasking shut down, the system hardware components are still nominally under the control of the OS, so you still have to allocate them before "hitting the hardware directly."

cd.device - ► *Is it still under OS control after system takeover?*

Yes.

► *How about when the CD is not being accessed?*

Yes, it is still nominally under the control of the OS.

Blitter -

► *Do we need to call OwnBlitter() and DisownBlitter() ?*

Yes this is recommended. After calling OwnBlitter() call WaitBlit() before using the blitter. Call DisownBlitter() before exiting.

► *Presumably cd.device uses the blitter?*

Currently it does not. This could change in a future version.

Audio -

► *For normal Paula audio, can we hit audio registers directly and ignore audio.device?*

No. Allocate all the channels you want to use at the highest priority (127). Then you may hit the hardware directly.

Interrupts -

► *Which ones does the OS have to have running?*

We do not want you to rely on such intimate information.

► *Can we override VBI or does OS need its VBI too?*

The OS needs its VBI. Stick with the Exec handler chain scheme.

► *How much overhead does the OS take?*

The OS interrupt routines take approximately 1/10 of 1% of CPU time.

Copper -

► *Can we install our own custom copper lists without calling those long-winded graphics.library functions?*

Custom copper lists will not work on the next generation of Amiga hardware so they should generally be avoided. If you must use custom copper lists, then before poking your custom copper list into the cop1/2jmp registers you should call the sequence: LoadView(NULL); WaitTOF(); WaitTOF(); to maximize compatibility with future systems. Future chip sets will probably have better (32 bit) coppers and the system will want to use these. If you poke into the old copper registers, the graphics.library will have no way of knowing and can't switch back to the 16-bit mode. Therefore the best thing to do is:

```
temp=GfxBase->ActiView;
LoadView(NULL);
WaitTOF();
WaitTOF();
...                /* custom.copllc = your stuff */

WaitTOF();
WaitTOF();
LoadView(temp);
custom.copllc = GfxBase->copint;
```

Also, keep in mind that the V39 OS has new graphic.library functions for performing operations that previously required a custom copper list in older versions of the Amiga OS.

For instance, V39 ChangeVPBitMap() offers very low overhead double-buffering for animation. ScrollVPort() has been rewritten and is 10x faster in V39 so that scrolling with 60hz displays is no problem for multiple ViewPorts on the CD machine (and A1200/A4000). The V39 LoadRGB32() function allows 256-color cycling animation in 60 Hz displays without problems in multiple ViewPorts. Also, the VideoControl() function has new tags VC_IntermediateCLUupdate and VTAG_IMMEDIATE can be used to improve performance or change some of the more esoteric features (color banking, sprite resolution, priorities) that previously required custom copper lists.

Remember: task priorities and the copper-interrupt server chain are your friends. You may not need a custom copper list. Consider using the new V39 functions instead.

Q5: *CD_EJECT command seems to have no purpose?*

Yes, you're right. This is present for compatibility reasons.

Q6: *Why bother with CD_MOTOR? What is gained by turning the motor off? The description of the CD_MOTOR command is a little confusing too. Surely, you don't recommend this is done often (or at all)?*

Indeed, there are very few good reasons to turn the CD motor off. Also, the CD_MOTOR Autodocs say "the spindle motor may automatically spin down if there is an absence of activity." This is in error. The system has no hidden "spin down." We will correct the docs.

Q7: *Since ReadJoyPort() allows for a mouse device, why is there no mouse support so that a mouse can be used with AmigaCD instead of the controller?*

There is mouse support. The ReadJoyPort() function returns a ULONG describing the type of the controller and its current state. If the type is JP_TYPE_MOUSE then the state information reflects the two mouse buttons (JPF_BTN1 and 2 are right and left respectively) and the vertical and horizontal mouse counter values (JP_MVERT_MASK and JP_MHORZ_MASK).

What stops us from using a mouse if we choose to leave input.device running?

Nothing. However, you don't need input.device. You can install a high-priority input handler or use ReadJoyPort() instead.

It looks like the game controller can emulate a mouse. Is this true?

The game controller can perform all the same functions as a mouse. However, it does not emulate the mouse at the hardware level.

Q8: *Does the game controller need input.device or can we kill input.device?*

If necessary, kill the input.device; the game controller does not need it. You can use SystemControlA() with the SCON_StopInput tag. Then use ReadJoyPort() to get controller information.

Q9: *We're unsure whether we have SimCD working properly. We seem to be able to call our application simply by CD'ing to the CD-partition on the hard drive and running the startup-sequence. (Admittedly, we are not using CD accesses yet.) Is this a side effect of setting up the CD-partition as an AmigaDOS device?*

The basic steps to using SimCD are as follows:

1. Set up a partition of the appropriate size on hard disk named CDN:.
2. Use ISOCD to create an ISO image of your programs and data files on the CDN: partition. You specify a source directory and a target partition. (Do not specify a file name for the image until you are ready to burn a gold disc.) Click on the Examine gadget and then the Build gadget. Exit.
3. Load SimCD. Under Configuration, select CDFS and cdtv.device only.
4. Set "Sim Device" to CDN: and hit the Simulate button.
5. SimCD sets up a simulated CD0: drive using the ISO image on CDN:.
6. Set your directory to CD0: and run your startup-sequence or executable.

Do we need SimCD when we actually use cd.device calls within our application?

No. SimCD currently emulates the older cdtv.device. We intend to upgrade SimCD, ISOCD and OPTCD in the future but they are not yet ready.

We also find that after running SimCD, we can no longer use ISOCD to alter the CD-partition. We have to delete the partition and then create a new partition. This is a pain and only happens when we run SimCD. Is this normal?

No. It is abnormal. When you want to change the image use ISOCD on CDN: (the partition name), not CD0: (the simulated CD drive).

Q10: *How do you use ISOCD to repeatedly alter your CD partition?*

The name of the partition you set up on hard disk to store the ISO image must *not* be named CD0:. This is reserved. Use CDN: instead. Then when you want to repeatedly use ISOCD, refer to CDN: (the partition name), not CD0: (the simulated CD drive).

Q11: *SystemControl() Autodoc states that the OS needs "crucial interrupts" What are these interrupts?*

There are currently 11 low-overhead interrupts per second taking less than 1/10 of 1% of CPU time. We prefer that you do not rely on (or alter) which particular interrupts are the "crucial ones."

Q12: *In CD_READ, what is the byte offset parameter?*

It is a byte offset from the beginning of the CD disc.

► How do you calculate this byte offset?

To read from a particular LSN (logical sector number), multiply the LSN by 2048 (there are 2048 bytes per sector, also known as a "frame"). Use this number for the byte offset parameter. For instance, the initial byte of LSN 1 would be byte offset 2048. The initial byte of LSN 2 would be byte offset 4096.

? Q13: How will gold discs be created?

In the U.K., send your ISO on DAT tape to the Maidenhead office, care of David Pocock. He will cut a disc for you. If it works, he will cut an additional disc for examination in West Chester. In the U.S., send your ISO on DAT to the West Chester office, care of Larry Feldman.

► Can we supply a DAT tape of an Amiga partition?

Yes, we prefer a DAT tape of your ISO file, although we will try to be as flexible as time permits. If you do not supply an ISO, you are adding time to the process (if this is the case, be sure to include the volume name for the CD-ROM disc).

? Q14: Is the mouse included with the expansion box kit?

Yes, we currently, we plan to include a mouse in the expansion box kit.

► What exactly will be included?

The current plan is to include a keyboard, floppy and mouse. However, be aware that the bill of materials may change prior to actual release.

? Q15: What companies are recommended for disc production?

For a complete list, see Chapter 2 of this document.

? Q16: What is going on with the different version numbers and names for calls?

We have recently upgraded the system from V39 to V40.

V39 = AmigaDOS 3.0 as shipped in Sept. 92.

V40 = AmigaDOS 3.01 with CD extensions for game machine.

? Q17: What exactly does `GetLanguageSelection()` return?

This was poorly documented. Sorry. The return values are:

```
#define GAME_LANG_UNKNOWN      0
#define GAME_LANG_AMERICAN    1
#define GAME_LANG_ENGLISH     2
#define GAME_LANG_GERMAN      3
#define GAME_LANG_FRENCH      4
#define GAME_LANG_SPANISH     5
```

```

#define GAME_LANG_ITALIAN      6
#define GAME_LANG_PORTUGUESE  7
#define GAME_LANG_DANISH      8
#define GAME_LANG_DUTCH       9
#define GAME_LANG_NORWEGIAN   10
#define GAME_LANG_FINNISH     11
#define GAME_LANG_SWEDISH     12
#define GAME_LANG_JAPANESE    13
#define GAME_LANG_CHINESE     14
#define GAME_LANG_ARABIC      15
#define GAME_LANG_GREEK       16
#define GAME_LANG_HEBREW      17
#define GAME_LANG_KOREAN      18

```

Q18: What should be the process of development for the machine?

1. Get a development system, an A4000 with a hard drive.
2. Develop or port game.
3. Test with ISOCD, SimCD and OPTCD on an A4000.
4. Transfer the ISO image onto DAT tape and send to Commodore in order to create a gold (master) disc. Or create the gold disc through other means.
5. Test gold disc on CD game machine.
6. Repeat steps 2-5 until ready for duplication.

What system is the suggested development system?

An Amiga 4000 with an 040 is a good development platform. The '040 allows you to use MMU-based debugging tools. The A4000 has the AA chip set. You should MAPROM the V40 developer Kickstart release to make the A4000 more like the game machine OS. MapROM also allows you to upgrade as new V40 Kickstarts are released (no ROMs or EPROMs required). The IDE disk drive gives a close approximation of CD-ROM speeds. For faster, larger hard drives, use a 2091 or 4091 controller and a SCSI drive. (The 2091 controller board must have the 7.0 ROM set to work properly with the A4000.)

Q19: Can you turn off the OS to get full control legally?

Yes. See the game.library function SystemControlA() used with the SCON_TakeOverSys tag allows you to suspend pre-emptive multitasking effectively giving you full control of the CPU. You should still use the system-approved method of getting control of the hardware resources that you want before you hit the hardware directly.

Q20: Are prototype Amiga CD³² game machines available for testing?

Early engineering samples of the Amiga CD³² game machine are available but there are very few of them to go around! (We appreciate your patience.) Sample units include the base system with CD-ROM drive, a special development-board that has Serial, Parallel, RGB, IDE and floppy connectors, and the game controller.

Will you be able to run off of a hard disk?

Yes with the development-board you can run your code and test it off of an IDE hard disk drive. With ISOCD, SimCD and OPTCD you can test the integrity of ISO images on hard disk (i.e., you don't have to cut a CD). However, be aware that the speeds are different (SimCD is slower than the actual CD will be) and low-level calls to `cd.device` will not work.

Will you be able to make CD calls?

You will not be able to make `cd.device` level calls to the ISO image on hard disk. However you can make `cd.device` calls to the CD-ROM.

Q21: Can you play music off of the CD and get data at the same time?

No. In general, this is a limitation of all current-generation CD systems. You can, however, load up Amiga audio data and start that playing in a loop while you load your data. Or you could load a compressed version of your executable files, then start playing CD audio after they are loaded, and then decompress your files and do other set up while the music plays. Be aware that if you do attempt to read CD data while CD audio is playing, the system will hang (instead of aborting the audio like the old `cdtv.device`). To stop CD audio playback, use `Exec AbortIO()` and `WaitIO()` calls.

Q22: Is it legal to use the blitter?

Yes it is. But please use `OwnBlitter()`, `WaitBlit()` and `DisownBlitter()` to get control of the hardware. This applies even if you have disabled multitasking with `SystemControl()`.

Q23: Can you use SNASM?

Currently, you cannot use SNASM because there are some problems with SNASM and the '020 processor used in the game system. Also there is no SCSI connector on the game machine or the development board.

Q24: Are there any calls which could guarantee that a program will be loaded to the same place in memory?

No, there are none.

I use SNASM and the code it generates is not relocatable (I think). What do you think?

In general, non-relocatable code will have problems on the game system. A custom loader might possibly solve the problem. There are other problems using SNASM with the game system because of its '020 processor and lack of SCSI interface.

If floppy drives are installed, will executables be loaded to a different memory location than they would be with no drives installed?

Yes. The floppy drives use a buffer and has other overhead that requires some memory. This is likely to cause a program to be run in a different memory location than it would on a system with no floppy drives.

? Q25: *Can you use some of the library calls to kill the OS?*

Basically yes. SystemControl() with the SCON_TakeOverSys tag disables task switching effectively giving you full control over the CPU. You will be in full control until you call Wait() or WaitPort() either directly or indirectly.

► Will the OS cause a bottleneck with the normal 300 Kb/sec throughput because of cycle-stealing?

Not if you disable task switching with SystemControl() and the SCON_TakeOverSys tag.

? Q26: *What is the best communication method for asking questions?*

The best method is to submit your questions to Commodore through an electronic BBS. In the U.K., Commodore maintains a closed conference on CIX. Send mail to cats_tech so that you can be added to the conference. In the U.S., Commodore uses the BIX network to support developers. Join amiga.dev/instructions for more information. Throughout Europe, Commodore supports developers on the ADSP network. Contact your local support manager to find out how to sign up. You can also FAX your question directly to CATS (0101-215-429-0643).

? Q27: *When the CD drive is in double-speed mode, how do you read the music and data files?*

You don't have to do anything special. The system handles it for you.

► Does the music automatically get handled?

Yes. For CD-DA reads the drive will automatically slow down to single-speed mode.

? Q28: *Will the game controller handle diagonals?*

Yes. The controller has a single omni-directional button that handles all 8 directions, similar to a joystick.

► Can the game controller emulate a mouse?

It can perform all the same functions as a mouse. However, it does not emulate the mouse at the hardware level. Furthermore, games that are optimized for mouse usage are likely to be hard to play on a system equipped only with a controller. If your title absolutely requires mouse emulation, CATS can supply you with an Amiga CD³²-compatible JoyMouse program for your startup-sequence.